МЕТОДИЧЕСКИЕ УКАЗАНИЯ
ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММНОГО МОДУЛЯ
ПМ.01 РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

для специальности

09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЕ

Базовая подготовка
среднего профессионального образования

2024

СОДЕРЖАНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические рекомендации по выполнению практических работ по программному модулю ПМ.01 Разработка модулей программного обеспечения для компьютерных систем составлены в соответствии с требованиями ФГОС СПО к минимуму содержания и уровню подготовки выпускников СПО по специальности 09.02.07 «Информационные системы и программирование» и на основе рабочей программы модуля.

Методические рекомендации включают практические занятия по четырем модулям:

МДК.01.01 Разработка программных модулей

МДК.01.02 Поддержка и тестирование программных модулей

МДК.01.03 Разработка мобильных приложений

МДК.01.04 Системное программирование

Требования к оформлению практических работ

Студент должен выполнить практическую работу в соответствии с полученным заданием. Каждый студент после выполнения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.

Отчет о проделанной работе следует выполнять на отдельных листах в клетку формата А4, которые хранятся в отдельных папках. Содержание отчета указано в описании практической работы.

Если студент не выполнил практическую работу или часть работы, то он может выполнить работу или оставшуюся часть во внеурочное время, согласованное с преподавателем.

Оценку по практической работе студент получает, с учетом срока выполнения работы, если:

- работа выполнена правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Зачет по практическим работам студент получает при условии выполнения всех предусмотренных программой работ, после сдачи отчетов по работам при получении удовлетворительных отметок.

Критерии оценки

Ответ оценивается отметкой «5», если:

работа выполнена полностью;

в логических рассуждениях и обосновании решения нет пробелов и ошибок;

в решении нет математических ошибок (возможны некоторые неточности, описки, которая не является следствием незнания или непонимания учебного материала).

Отметка «4» ставится в следующих случаях:

работа выполнена полностью, но обоснования шагов решения недостаточны (если умение обосновывать рассуждения не являлось специальным объектом проверки);

допущены одна ошибка, или есть два – три недочёта в выкладках, рисунках, чертежах или графиках (если эти виды работ не являлись специальным объектом проверки).

Отметка «3» ставится, если:

допущено не более двух ошибок или более двух – трех недочетов в выкладках, чертежах или графиках, но обучающийся обладает обязательными умениями по проверяемой теме.

Отметка «2» ставится, если:

допущены существенные ошибки, показавшие, что обучающийся не обладает обязательными умениями по данной теме в полной мере.

Преподаватель может повысить отметку за оригинальный ответ на вопрос или оригинальное решение задачи, которые свидетельствуют о высоком математическом развитии обучающегося; за решение более сложной задачи или ответ на более сложный вопрос, предложенные обучающемуся дополнительно после выполнения им каких-либо других заданий.

Практические занятия по модулям

МДК.01.01 Разработка программных модулей

Практическая работа №1 Оценка сложности алгоритмов сортировки

Цель работы: изучить основные алгоритмы внутренних сортировок и научиться решать задачи сортировок массивов различными методами (бинарная *пирамидальная сортировка*, метод Шелла, быстрая *сортировка* Хоара, *сортировка слиянием*).

При выполнении лабораторной работы для каждого задания требуется написать программу на языке С++, которая получает на входе числовые данные, выполняет генерацию и *вывод* массива *указанного типа* в зависимости от постановки задачи. В каждой задаче необходимо выполнить сортировку данных и реализовать один из алгоритмов: бинарной *пирамидальной сортировки*, сортировки по методу Шелла, быстрой сортировки Хоара и *сортировки слиянием* в виде отдельных функций. Ввод данных осуществляется с клавиатуры или из файла с учетом требований к *входным данным*, содержащихся в постановке задачи. Ограничениями на *входные данные* является *диапазон* используемого *числового типа* данных в языке С++ и максимально допустимый размер объявляемого одномерного массива.

Теоретические сведения.

Ознакомьтесь с материалом лекции 42.

Задания к лабораторной работе.

Выполните приведенные ниже задания.

1. На основании приведенных в лекции 42 функций реализуйте алгоритмы внутренних сортировок.
2. Даны два целочисленных файла, упорядоченных по возрастанию. Сформировать третий файл на основе данных, который также упорядочен и представляет операцию с элементами исходных файлов:
   - объединение (содержит числа, принадлежащие хотя бы одному из множеств);
   - перечисление (числа, принадлежащие обоим множествам);
   - разность (числа, принадлежащие первому множеству, но не второму);
   - симметричную разность (объединение *разностей множеств*).
3. Заданы N (N<=5000) попарно различных длин отрезков. Вычислить количество способов, которыми из отрезков можно сложить треугольник.
4. Дана целочисленная квадратная матрица размером n. Упорядочить значения так, чтобы $a_{11}<=a_{12}<=<=a_{1n}<=a_{21}<=a_{22}<=<=a_{2n}<=<=a_{n1}<=a_{n2}<=<=a_{nn}$.
5. Дан целочисленный массив. Выполните проверку уникальности. Удалите из массива повторные вхождения чисел.

Указания к выполнению работы.

Каждое задание необходимо решить в соответствии с изученными алгоритмами внутренних сортировок: бинарной *пирамидальной сортировки*, сортировки по методу Шелла, быстрой сортировки Хоара и *сортировки слиянием*. Программные коды следует реализовать на языке С++. Рекомендуется воспользоваться материалами лекции 42, где подробно рассматриваются описание используемых в работе алгоритмов, примеры их реализации на языке С++. Программу для решения каждого задания необходимо разработать методом процедурной абстракции, используя функции. Этапы решения сопроводить комментариями в коде. В отчете следует отразить разработку и обоснование математической модели решения задачи и примеры входных и выходных файлов.

Следует реализовать каждое *задание в соответствии* с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать *метод решения* задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке С++;

- разработать контрольный тест к программе;
- отладить программу;
- представить отчет по работе.

Требования к отчету.

Отчет по лабораторной работе должен соответствовать следующей структуре.

- Титульный лист.
- Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.
- *Математическая модель*. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
- Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается *абстракция данных*, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).
- *Листинг программы*. Подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio 2010.
- Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
- Выводы по лабораторной работе.
- Ответы на контрольные вопросы.

Контрольные вопросы

1. Чем можно объяснить многообразие алгоритмов сортировок?
2. Почему на данный момент не существует универсального алгоритма сортировки?
3. Как соблюдение свойств устойчивости и естественности влияет на трудоемкость алгоритма сортировки?
4. За счет чего в алгоритмах быстрых сортировок происходит выигрыш при выполнении операций сравнения и перестановок?
5. Какие из перечисленных алгоритмов наиболее эффективны на почти отсортированных массивах: бинарная *пирамидальная сортировка*, *сортировка слиянием*, сортировка Шелла и сортировка Хоара? За счет чего происходит выигрыш?
6. Почему алгоритмы быстрых сортировок не дают большого выигрыша при малых размерах массивов?
7. В чем преимущества и недостатки по отношению друг к другу следующих алгоритмов сортировок: бинарная *пирамидальная сортировка*, *сортировка слиянием*, сортировка Шелла и сортировка Хоара?
8. Как определить, какому алгоритму сортировки отдать предпочтение при решении задачи?

Практическая работа №2 Оценка сложности алгоритмов поиска

Цель работы: изучить основные алгоритмы поиска

Теоретические сведения

Правильность — далеко не единственное качество, которым должна обладать хорошая *программа*. Одним из важнейших является эффективность, характеризующая прежде всего *время выполнения* программы $T(n)$ для различных входных данных (параметра n).

Нахождение точной зависимости $T(n)$ для конкретной программы — задача достаточно сложная. *По* этой причине обычно ограничиваются **асимптотическими оценками** этой функции, то есть описанием ее примерного поведения при больших значениях параметра n. Иногда для асимптотических оценок используют традиционное *отношение* O (читается "О большое") между двумя функциями $f(n) = O(g(n))$, *определение* которого можно найти в любом учебнике *по* математическому анализу, хотя чаще применяют *отношение эквивалентности*

Θ (читается "тэта большое"). Его формальное *определение* есть, например, в книге [8], хотя нам пока достаточно будет понимания данного вопроса в общих чертах.

В качестве первого примера вернемся к только что рассмотренным программам нахождения факториала числа. Легко видеть, что количество операций, которые должны быть выполнены для нахождения факториала n ! числа n в первом приближении прямо пропорционально этому числу, ибо количество повторений *цикла* (итераций) в данной программе равно n . В подобной ситуации принято говорить, что *программа* (или *алгоритм*) имеет ***линейную сложность*** (сложность $O(n)$ или $\Theta(n)$).

Можно ли вычислить *факториал* быстрее? Оказывается, да. Можно написать такую программу, которая будет давать правильный результат для тех же значений n , для которых это делают все приведенные выше программы, не используя при этом ни итерации, ни рекурсии. Ее сложность будет $\Theta(1)$, что фактически означает организацию вычислений *по* некоторой формуле без применения циклов и рекурсивных вызовов!

Не менее интересен и пример вычисления n -го *числа Фибоначчи*. В процессе ее исследования фактически уже было выяснено, что ее сложность является *экспоненциальной* и равна $\Theta(2^n)$. Подобные программы практически не применимы на практике. В этом очень легко убедиться, попробовав вычислить с ее помощью 40-е *число Фибоначчи*. *По* этой причине вполне актуальна следующая задача.

Задача 5.4. Напишите программу, печатающую n -ое *число Фибоначчи*, которая имела бы *линейную сложность*.

Вот решение этой задачи, в котором переменные j и k содержат значения двух последовательных чисел Фибоначчи.

Текст программы

```
public class FibIv1 {
    public static void main(String[] args) throws Exception {
      int n = Xterm.inputInt("Введите n -> ");
      Xterm.print("f(" + n + ")");
      if (n < 0) {
        Xterm.print(" не определено\n");
      }  else if (n < 2)
        { Xterm.println(" = " +
        n);
      } else {
        long i = 0;
        long j = 1;
        long k;
        int  m = n;
        while (--m > 0) {
          k = j;
          j += i;
          i = k;
        }
        Xterm.println(" = " + j);
      }
    }
}
```

Следующий вопрос вполне естественен — а можно ли находить *числа Фибоначчи* еще быстрее?

После изучения определенных разделов математики совсем просто вывести следующую формулу для n -ого *числа Фибоначчи* f_n, которую легко проверить для небольших значений n:

$$f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right].$$

Может показаться, что основываясь на ней, легко написать программу со сложностью $\Theta(1)$, не использующую итерации или рекурсии.

Текст программы

```
public class FibIv2 {
    public static void main(String[] args) throws Exception {
       int    n = Xterm.inputInt("Введите n -> ");
       double f = ( 1.0 + Math.sqrt(5.) ) / 2.0;
       int    j = (int)( Math.pow(f,n) / Math.sqrt(5.) + 0.5 );

       Xterm.println("f(" + n + ") = " + j);
    }
}
```

На самом деле эта *программа* использует *вызов функции* возведения в степень { Math.pow(f,n) }, которая не может быть реализована быстрее, чем за логарифмическое время ($\log_2 n$). Про алгоритмы, в которых количество операций примерно пропорционально $\log n$ (в информатике принято не указывать *основание* двоичного логарифма) говорят, что они имеет ***логарифмическую сложность*** ($\Theta(\log n)$).

Для вычисления n -го *числа Фибоначчи* существует такой *алгоритм*, программную реализацию которого мы приведем без дополнительных комментариев, — иначе нужно объяснять слишком много (*связь* чисел Фибоначчи со степенями некоторой матрицы порядка два, использование классов для работы с матрицами, *алгоритм* быстрого возведения матрицы в степень).

Задача 5.5. Напишите программу, печатающую n -ое *число Фибоначчи*, которая имела бы *логарифмическую сложность*.

Текст программы

```
public class FibIv3 {
    public static void main(String[] args) throws Exception {
       int n = Xterm.inputInt("Введите n -> ");
       Xterm.print("f(" + n + ")");
       if (n < 0) {
          Xterm.println(" не определено");
       }   else   if   (n   <   2)
          { Xterm.println("  =  " +
          n);
       } else {
          Matrix b = new Matrix(1, 0, 0, 1);
          Matrix c = new Matrix(1, 1, 1, 0);
          while (n>0) {
             if ((n&1) == 0) {
                n >>>= 1; c.square();
             } else {
                n  -=  1; b.mul(c);
             }
          }
          Xterm.println(" = " + b.fib());
       }
    }
}
class Matrix {
    private long a, b, c, d;

    public Matrix(long a, long b, long c, long d)
       { this.a = a; this.b = b; this.c = c; this.d = d;
```

```
    }
    public void mul(Matrix m) {
       long a1 = a*m.a+b*m.c;  long b1 = a*m.b+b*m.d;
       long c1 = c*m.a+d*m.c;  long d1 = c*m.b+d*m.d;
       a = a1; b = b1; c = c1; d = d1;
    }
    public    void    square()    {
       mul(this);
    }
    public     long      fib()
       { return b;
    }
}
```

Если попробовать посчитать десятимиллионное *число Фибоначчи* с помощью этой и предыдущей программ, то разница во времени счета будет вполне очевидной. К сожалению, результат будет неверным (в обоих случаях) в силу ограниченности диапазона чисел типа long.

В заключение приведем сравнительную таблицу времен выполнения алгоритмов с различной сложностью и объясним, почему с увеличением быстродействия компьютеров важность использования быстрых алгоритмов значительно возрастает.

Рассмотрим четыре алгоритма решения одной и той же задачи, имеющие сложности $\log n$, n , n^2 и 2^n соответственно. Предположим, что второй из этих алгоритмов требует для своего выполнения на некотором компьютере при значении параметра $n = 10^3$ ровно одну минуту времени. Тогда времена выполнения всех этих четырех алгоритмов на том же компьютере при различных значениях параметра будут примерно такими, как в таблице 5.1.

Таблица 5.1. Сравнительная таблица времен выполнения алгоритмов

Сложность алгоритма	**n=10**	**n=10^3**	**n=10^6**
log n	0.2 сек.	0.6 сек.	1.2 сек.
n	0.6 сек.	1 мин.	16.6 час.
n^2	6 сек.	16.6 час.	1902 года
2^n	1 мин.	10^{295} лет	10^{300000} лет

Когда начинающие программисты тестируют свои программы, то значения параметров, от которых они зависят, обычно невелики. Поэтому даже если при написании программы был применен неэффективный *алгоритм*, это может остаться незамеченным. Однако, если подобную программу попытаться применить в реальных условиях, то ее практическая непригодность проявится незамедлительно.

С увеличением быстродействия компьютеров возрастают и значения параметров, для которых работа того или иного алгоритма завершается за приемлемое время. Таким образом, увеличивается среднее *значение* величины n , и, следовательно, возрастает величина отношения времен выполнения быстрого и медленного алгоритмов. **Чем быстрее компьютер, тем больше относительный проигрыш при использовании плохого алгоритма**!

Практическая работа №3 Оценка сложности рекурсивных алгоритмов

Цель работы: *Получение практических навыков решения задач с использованием рекурсивных алгоритмов.*

1. Методика выполнения работы.

Составить программу, которая вычисляет значения заданной функции для заданных значений X – аргумента с заданной точностью ε и выводит значения аргумента и функции в табличной форме. При программировании вычисления значений функции использовать рекуррентную формулу и рекурсивные функции.

Программа должна включать:

1. ввод исходных данных (с клавиатуры и из файла);
2. функцию вычисления очередного члена ряда с использованием рекуррентной формулы;
3. из функции для вычисления очередного члена ряда вызывать другие, в том числе рекурсивные функции, например для вычисления степени X, факториала и пр.;
4. для вычисления суммы ряда использовать 3 разные функции, использующие:
   - оператор While;
   - оператор Repeat – Until;
   - рекурсивное суммирование;
5. вывод результатов выполнения программы (в процессе отладки программы – на экран; после отладки – в файл);
6. для управления работой программой разработать структуру меню для вызова каждой процедуры (формирование меню осуществляется средствами модуля CRT);
7. для тестирования программы сформировать исходные данные таким образом, чтобы проверить каждый вариант альтернативы каждого разветвления алгоритма.

Для выполнения работы необходимо знание следующих теоретических вопросов из курса предмета «Основы алгоритмизации и программирования»:

- строение функции и правила ее вызова;
- определение рекуррентной формулы;
- определение рекурсивной функции и ее строение;
- приемы отладки рекурсивных функций;
- работа с файлами.

2. Теоретическая часть.

Рекурсия - это одна из фундаментальных концепций в математике и программировании и является мощным средством, позволяющим строить элегантные и выразительные алгоритмы. ***Рекурсия*** — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается *сама к себе.* Если процедура p содержит явное обращение к самой себе, то она называется *явно рекурсивной*. Если процедура p содержит обращение к некоторой процедуре q, которая в свою очередь содержит прямое или косвенное обращение к p, то p - называется *косвенно рекурсивной.*

Но рекурсивная программа не может вызывать себя бесконечно, иначе она никогда не остановится, таким образом в программе (функции) должен присутствовать еще один важный элемент - так называемое терминальное (граничное) условие, то есть условие при котором программа прекращает рекурсивный процесс.

Вот типичная конструкция такого рода:

```
procedure     proc(i:integer);
begin
operation1;
```

```
if logb then proc(i+1); operation2;
end;
```

Вызов proc(1) означает, что proc вызывает себя раз за разом с помощью proc(2), proc(3),.. до тех пор, пока условие logb не отменит новый вызов. При каждом вызове выполняется оператор operation1, после чего порядок выполнения операторов прерывается новым вызовом proc(i+1).
В Паскале можно пользоваться именами лишь тогда, когда в тексте программы этому предшествует их описание. Рекурсия является единственным исключением из этого правила. Имя proc можно использовать сразу же, не закончив его описания.
Рекурсия не должна восприниматься как некий программистский трюк. Это скорее некий принцип, метод. Если в программе нужно выполнить что-то повторно, можно действовать двумя способами:

- с помощью последовательного присоединения (или итерации в форме цикла);
- с помощью вложения одной операции в другую (а именно, рекурсий).

Рассмотрим на примере принципиальное различие между итерацией и рекурсией. Итерации необходим цикл и локальная переменная k как переменная цикла. Рекурсии ничего этого не требуется!

```
program iterativ_zu_rekursion;
var n:integer;
procedure rekursion (i:integer);
begin
writeln(i:30);
if i < 1 then rekursion(i-1);
writeln(i:3);
end; (* Рекурсия *)

procedure schleife(i:integer);
var k:integer;
bagin k
:=1;
while k <= i do begin
write(k:3);
k :=k+1;
end;
end; (* Цикл *)

begin
write('Введите n:'); readln(n);
writeln('Пока:');
scheife(n); writeln;
writeln('Рекурсия'); rekursion(n);
end.
```

Реккурентность - это рекурсивное определение функции. Они широко распространены в математике. Наиболее знакомая из такого рода функций - это факториал. Факториал - это произведение натуральных чисел от единицы до какого - либо данного натурального числа. Он определяется формулой:

N!=N((N-1)!, для N>=1 и 0! = 1.
Это напрямую соответствует нижеследующей рекурсивной программе:

```
function factorial( N : integer ) : integer;
begin
if N=0 then factorial := 1
else
factorial := N * factorial(N-1);
end;
```

Эта программа демонстрирует основные свойства рекурсивных программ: программа вызывает сама себя (с меньшим значением аргумента), и у нее есть граничное условие при котором она прямо вычисляет результат.
Необходимо также помнить о том, что это - программа, а не формула: например ни формула, ни программа не работают с отрицательными N, но губительные последствия попытки произвести вычисления для отрицательного числа более заметны для программы, чем для формулы. Вызов factorial(-1) приведет к бесконечному рекурсивному циклу. Поэтому перед вызовом данной программы нужно делать проверку условия неотрицательности.
Обращение к рекурсивной подпрограмме ничем не отличается от вызова любой другой подпрограммы. При этом при каждом новом рекурсивном обращении в памяти создаётся новая копия подпрограммы со всеми локальными переменными. Такие копии будут порождаться до выхода на граничное условие. Очевидно, в случае отсутствия граничного условия, неограниченный рост числа таких копий приведёт к аварийному завершению программы за счёт переполнения стека.
Порождение все новых копий рекурсивной подпрограммы до выхода на граничное условие называется *рекурсивным спуском*. Максимальное количество копий рекурсивной подпрограммы, которое одновременно может находиться в памяти компьютера, называется *глубиной рекурсии*. Завершение работы рекурсивных подпрограмм, вплоть до самой первой, инициировавшей рекурсивные вызовы, называется *рекурсивным подъёмом*.
Выполнение действий в рекурсивной подпрограмме может быть организовано одним из вариантов:

```
Begin Begin Begin
P; операторы; операторы;
операторы; P P;
End; End; операторы End;
```

Вариант 1 реализует рекурсивный подъём, вариант 2 - рекурсивный спуск и вариант 3 - рекурсивный спуск, и рекурсивный подъём. Здесь P — рекурсивная подпрограмма. Как видно из примера, действия могут выполняться либо на одном из этапов рекурсивного обращения, либо на обоих сразу. Способ организации действий диктуется логикой разрабатываемого алгоритма.

Различие в написании рекурсивных определений в виде функций и процедур рассмотрим на примере вычисления факториала.

```
{Функция}
Function Factorial(N:integer):Extended;
Begin
If N<=1 Then Factorial:=1
Else Factorial:=Factorial(N-1)* N End;
```

```
{Процедура}
Procedure Factorial(N:integer; Var F:Extended);
Begin
If N<=1 Then F:=1
Else
Begin
Factorial(N-1, F);
F:=F*N
End;
End;
```

В приведенных выше примерах программ действия выполняются на рекурсивном подъёме.

Рассмотрим еще один пример: *Вычислить сумму элементов линейного массива.*

При решении задачи используем следующее соображение: сумма равна нулю, если количество элементов равно нулю, и сумме всех предыдущих элементов плюс последний, если количество элементов не равно нулю.

```
Program Rec2;
Type LinMas = Array[1..100] Of Integer;
Var A : LinMas;
I, N : Byte;
{Рекурсивная функция}
Function Summa(N : Byte; A: LinMas) : Integer;
Begin
If N = 0 Then Summa := 0
Else Summa := A[N] + Summa(N - 1, A)
End;
{Основная программа}
Begin
Write('Количество элементов массива? ');
ReadLn(N);
Randomize;
For I := 1 To N Do
Begin
A[I] := -10 + Random(21); Write(A[I] : 4)
End;
WriteLn;
WriteLn('Сумма: ', Summa(N, A)) End.
```

Подводя итог, заметим, что использование рекурсии является красивым приёмом программирования. Краткость и выразительность большинства рекурсивных процедур упрощает их чтение и сопровождение. В то же время в большинстве практических задач этот приём неэффективен с точки зрения расходования таких ресурсов ЭВМ, как память и время исполнения программы. Использование рекурсии увеличивает время исполнения программы и зачастую требует значительного объёма памяти для хранения копий подпрограммы на рекурсивном спуске. Поэтому на практике выбор между рекурсивным и итерационным алгоритмами зависит от постановки задачи и определения критериев ее эффективности.

Практическая работа №5 Работа с классами.

C# является полноценным объектно-ориентированным языком. Это значит, что программу на C# можно представить в виде взаимосвязанных взаимодействующих между собой объектов.

Описанием объекта является **класс**, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, какие-то другие характеристики. То есть некоторый шаблон - этот шаблон можно назвать классом. Конкретное воплощение этого шаблона может отличаться, например, одни люди имеют одно имя, другие - другое имя. И реально существующий человек (фактически экземпляр данного класса) будет представлять объект этого класса.

По умолчанию проект консольного приложения уже содержит один класс Program, с которого и начинается выполнение программы.

По сути класс представляет новый тип, который определяется пользователем. Класс определяется с помощью ключевого слова **class**:

```
class Person
{

}
```

Где определяется класс? Класс можно определять внутри пространства имен, вне пространства имен, внутри другого класса. Как правило, классы помещаются в отдельные файлы. Но в данном случае поместим новый класс в файле, где располагается класс Program. То есть файл Program.cs будет выглядеть следующим образом:

```
using System;

namespace HelloApp
{
class Person
{

    }
class Program
{
static void Main(string[] args)
{

        }
    }
}
```

Вся функциональность класса представлена его членами - полями (полями называются переменные класса), свойствами, методами, событиями. Например, определим в классе Person поля и метод:

```
using System;

namespace HelloApp
{
class Person
{
public string name; // имя
public int age = 18;            // возраст

public void GetInfo()
```

```
        {
            Console.WriteLine($"Имя: {name}  Возраст: {age}");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Person tom;
        }
    }
}
```

В данном случае класс Person представляет человека. Поле name хранит имя, а поле age - возраст человека. А метод GetInfo выводит все данные на консоль. Чтобы все данные были доступны вне класса Person переменные и метод определены с модификатором public. Поскольку поля фактически те же переменные, им можно присвоить начальные значения, как в случае выше, поле age инициализировано значением 18.

Так как класс представляет собой новый тип, то в программе мы можем определять переменные, которые представляют данный тип. Так, здесь в методе Main определена переменная tom, которая представляет класс Person. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение **null**. Поэтому вначале необходимо создать объект класса Person.

Конструкторы

Кроме обычных методов в классах используются также и специальные методы, которые называются **конструкторами**. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

Конструктор по умолчанию

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор по умолчанию. Такой конструктор не имеет параметров и не имеет тела.

Выше класс Person не имеет никаких конструкторов. Поэтому для него автоматически создается конструктор по умолчанию. И мы можем использовать этот конструктор. В частности, создадим один объект класса Person:

```
class Person
{
public string name; // имя
public int age;           // возраст

public void GetInfo()
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
        }
    }
class Program
{
static void Main(string[] args)
{
Person tom = new Person();
tom.GetInfo();              // Имя: Возраст: 0

tom.name = "Tom";
tom.age = 34;
```

```
20          tom.GetInfo();  // Имя: Tom Возраст: 34
21          Console.ReadKey();
22      }
23  }
```

Для создания объекта Person используется выражение new Person(). Оператор **new** выделяет память для объекта Person. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта Person. А переменная tom получит ссылку на созданный объект.

Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа string и классов - это значение **null** (то есть фактически отсутствие значения).

После создания объекта мы можем обратиться к переменным объекта Person через переменную tom и установить или получить их значения, например, tom.name = "Tom";.

Консольный вывод данной программы:

```
Имя:       Возраст: 0
Имя: Tom           Возраст: 34
```

Создание конструкторов

Выше для инициализации объекта использовался конструктор по умолчанию. Однако мы сами можем определить свои конструкторы:

```
1   class Person
2   {
3   public string name;
4   public int age; 5
6   public Person() { name = "Неизвестно"; age = 18; }          // 1 конструктор
7
8           public Person(string n) { name = n; age = 18; }       // 2 конструктор
9
10          public Person(string n, int a) { name = n; age = a; }  // 3
конструктор 11
12  public void GetInfo()
13  {
14  Console.WriteLine($"Имя: {name}  Возраст: {age}");
15          }
16      }
```

Теперь в классе определено три конструктора, каждый из которых принимает различное количество параметров и устанавливает значения полей класса. Используем эти трукто-консры:

```
1       static void Main(string[] args)
2       {
3           Person tom = new Person();          // вызов 1-ого конструктора без параметров
4           Person bob = new Person("Bob");     //вызов 2-ого конструктора с одним параметром
5           Person sam = new Person("Sam", 25); // вызов 3-его конструктора с двумя
6       параметрами
7
8
9           bob.GetInfo();          // Имя: Bob  Возраст: 18
10          tom.GetInfo();          // Имя: Неизвестно  Возраст:
11          18 sam.GetInfo();       // Имя: Sam  Возраст: 25
        }
```

Консольный вывод данной программы:

```
Имя: Неизвестно  Возраст: 18
Имя: Bob  Возраст: 18
Имя: Sam  Возраст: 25
```

При этом если в классе определены конструкторы, то при создании объекта необходимо использовать один из этих конструкторов.

Стоит отметить, что начиная с версии C# 9.0 мы можем сократить вызов конструктора, убрав из него название типа:

```
Person tom = new ();              // аналогично new Person();
Person bob = new ("Bob");         // аналогично new Person("Bob");
Person sam = new ("Sam", 25);   // аналогично new Person("Sam", 25);
```

Ключевое слово this

Ключевое слово **this** представляет ссылку на текущий экземпляр класса. В каких ситуациях оно нам может пригодиться? В примере выше определены три конструктора. Все три конструктора выполняют однотипные действия - устанавливают значения полей name и age. Но этих повторяющихся действий могло быть больше. И мы можем не дублировать функциональность конструкторов, а просто обращаться из одного конструктора к другому через ключевое слово this, передавая нужные значения для параметров:

```
class Person
{
public string name;
public int age; 5
public Person() : this("Неизвестно")
{
}
public Person(string name) : this(name, 18)
{
}
public Person(string name, int age)
{
this.name = name;
this.age = age;
        }
public void GetInfo()
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
        }
    }
```

В данном случае первый конструктор вызывает второй, а второй конструктор вызывает третий. По количеству и типу параметров компилятор узнает, какой именно конструктор вызывается. Например, во втором конструкторе:

```
public Person(string name) : this(name, 18)
{
}
```

идет обращение к третьему конструктору, которому передаются два значения. Причем в начале будет выполняться именно третий конструктор, и только потом код второго конструктора.

Также стоит отметить, что в третьем конструкторе параметры называются также, как и поля класса.

```
public Person(string name, int age)
{
```

```
    this.name = name;
    this.age = age;
}
```

И чтобы разграничить параметры и поля класса, к полям класса обращение идет через ключевое слово this. Так, в выражении this.name = name; первая часть this.name означает, что name - это поле текущего класса, а не название параметра name. Если бы у нас параметры и поля назывались по-разному, то использовать слово this было бы необязательно. Также через ключевое слово this можно обращаться к любому полю или методу.

Инициализаторы объектов

Для инициализации объектов классов можно применять **инициализаторы**. Инициализаторы представляют передачу в фигурных скобках значений доступным полям и свойствам объекта:

```
Person tom = new Person { name = "Tom", age=31 };
tom.GetInfo();          // Имя: Tom  Возраст: 31
```

С помощью инициализатора объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.

При использовании инициализаторов следует учитывать следующие моменты:

- С помощью инициализатора мы можем установить значения только доступных из внешнего кода полей и свойств объекта. Например, в примере выше поля name и age имеют модификатор доступа public, поэтому они доступны из любой части программы.
- Инициализатор выполняется после конструктора, поэтому если и в конструкторе, и в инициализаторе устанавливаются значения одних и тех же полей и свойств, то значения, устанавливаемые в конструкторе, заменяются значениями из инициализатора.

Практическая работа №6 Перегрузка методов

Иногда возникает необходимость создать один и тот же метод, но с разным набором параметров. И в зависимости от имеющихся параметров применять определенную версию метода. Такая возможность еще называется **перегрузкой методов** (method overloading).

И в языке C# мы можем создавать в классе несколько методов с одним и тем же именем, но разной сигнатурой. Что такое сигнатура? **Сигнатура** складывается из следующих аспектов:

Имя метода

Количество параметров

Типы параметров

Порядок параметров

Модификаторы параметров

Но названия параметров в сигнатуру НЕ входят. Например, возьмем следующий метод:

```
public int Sum(int x, int y)
{
return x + y;
}
```

У данного метода сигнатура будет выглядеть так: Sum(int, int)

И перегрузка метода как раз заключается в том, что методы имеют разную сигнатуру, в которой совпадает только название метода. То есть методы должны отличаться по:

Количеству параметров

Типу параметров

Порядку параметров

Модификаторам параметров Например, пусть у нас есть следующий класс:

```
class Calculator
{
```

```
    public void Add(int a, int b)
    {
        int result = a + b;
        Console.WriteLine($"Result is {result}");
    }
    public void Add(int a, int b, int c)
    {
        int result = a + b + c;
        Console.WriteLine($"Result is {result}");
    }
    public int Add(int a, int b, int c, int d)
    {
        int result = a + b + c + d;
        Console.WriteLine($"Result is {result}");
        return result;
    }
    public void Add(double a, double b)
    {
        double result = a + b;
        Console.WriteLine($"Result is {result}");
    }
}
```

Здесь представлены четыре разных версии метода Add, то есть определены четыре перегрузки данного метода.

Первые три версии метода отличаются по количеству параметров. Четвертая версия совпадает с первой по количеству параметров, но отличается по их типу. При этом достаточно, чтобы хотя бы один параметр отличался по типу. Поэтому это тоже допустимая перегрузка метода Add.

То есть мы можем представить сигнатуры данных методов следующим образом:

```
Add(int, int)
Add(int, int, int)
Add(int, int, int, int)
Add(double, double)
```

После определения перегруженных версий мы можем использовать их в программе:

```
class Program
{
static void Main(string[] args)
{
Calculator calc = new Calculator();
calc.Add(1, 2); // 3
calc.Add(1, 2, 3); // 6
calc.Add(1, 2, 3, 4); // 10
calc.Add(1.4, 2.5); // 3.9

            Console.ReadKey();
        }
    }
```

Консольный вывод:

```
Result is 3
Result is 6
Result is 10
Result is 3.9
```

Также перегружаемые методы могут отличаться по используемым модификаторам. Например:

```
1    void Increment(ref int val)
2    {
3    val++;
4    Console.WriteLine(val);
5        }
6
7    void Increment(int val)
8    {
9    val++;
10   Console.WriteLine(val);
11       }
```

В данном случае обе версии метода Increment имеют одинаковый набор параметров одинакового типа, однако в первом случае параметр имеет модификатор ref. Поэтому обе версии метода будут корректными перегрузками метода Increment.

А отличие методов по возвращаемому типу или по имени параметров не является основанием для перегрузки. Например, возьмем следующий набор методов:

```
1    int Sum(int x, int y)
2    {
3    return x + y;
4        }
5    int Sum(int number1, int number2)
6    {
7    return x + y;
8        }
9    void Sum(int x, int y)
10   {
11   Console.WriteLine(x + y);
12       }
```

Сигнатура у всех этих методов будет совпадать:

```
1    Sum(int, int)
```

Поэтому данный набор методов не представляет корректные перегрузки метода Sum и работать не будет.

Практическая работа №7 Определение операций в классе

Нередко различные классы и структуры оформляются в виде отдельных библиотек, которые компилируются в файлы dll и затем могут подключать в другие проекты. Благодаря этому мы можем определить один и тот же функционал в виде библиотеки классов и подключать в различные проекты или передавать на использование другим разработчикам.

Создадим и подключим библиотеку классов.

Возьмем имеющийся проект консольного приложения .NET Core, например, созданный в прошлых темах. В структуре проекта нажмем правой кнопкой на название решения и далее в появившемся контекстном меню выберем **Add -> New Project...** (Добавить новый проект):

Далее в списке шаблонов проекта найдем пункт **Class Library (.NET Core)**:

Затем дадим новому проекту какое-нибудь название, например, MyLib:

После этого в решение будет добавлен новый проект, в моем случае с названием MyLib:

По умолчанию новый проект имеет один пустой класс Class1 в файле Class1.cs. Мы можем этот файл удалить или переименовать, как нам больше нравится.
Например, переименуем файл Class1.cs в Person.cs, а класс Class1 в Person. Определим в классе Person простейший код:

```
1    public class Person
2    {
3    public string name;
4    public int age;
5        }
```

Теперь скомпилируем библиотеку классов. Для этого нажмем правой кнопкой на проект библиотеки классов и в контекстном меню выберем пункт **Rebuild**:

После компиляции библиотеки классов в папке проекта в каталоге *bin/Debug/netcoreapp3.0* мы сможем найти скомпилированный файл dll (MyLib.dll). Подключим его в основной проект. Для этого в основном проекте нажмем правой кнопкой на узел **Dependencies** и в контекстном меню выберем пункт **Add Reference**:

Далее нам откроется окно для добавления библиотек. В этом окне выберем пункт Solution,который позволяет увидеть все библиотеки классов из проектов текущего решения, поставим отметку рядом с нашей библиотекой и нажмем на кнопку OK:

Если наша библиотека вдруг представляет файл dll, который не связан ни с каким проектом в нашем решении, то с помощью кнопки **Browse** мы можем найти местоположение файла dll и также его подключить.
После успешного подключения библиотеки в главном проекте изменим класс Program, чтобы он использовал класс Person из библиотеки классов:

```
1    using System;
2    using MyLib;       // подключение пространства имен из библиотеки классов
3
4    namespace HelloApp
5    {
6    class Program
7    {
8    static void Main(string[] args)
9    {
10   Person tom = new Person { name = "Tom", age = 35 };
11   Console.WriteLine(tom.name);
12               }
13           }
14       }
```

Практическая работа №8 Создание наследованных классов

Наследование (inheritance) является одним из ключевых моментов ООП. Благодаря наследованию один класс может унаследовать функциональность другого класса.
Пусть у нас есть следующий класс Person, который описывает отдельного человека:

```
1    class Person
2    {
3    private string _name; 4
5    public string Name
6    {
7    get { return _name; }
8    set { _name = value; }
9            }
10   public void Display()
11   {
12   Console.WriteLine(Name);
13           }
14       }
```

Но вдруг нам потребовался класс, описывающий сотрудника предприятия - класс Employee. Поскольку этот класс будет реализовывать тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником, или подклассом) от класса Person, который, в свою очередь, называется базовым классом или родителем (или суперклассом):

```
1    class Employee : Person
2    {
3
4        }
```

После двоеточия мы указываем базовый класс для данного класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же свойства, методы, поля, которые есть в классе Person. Единственное, что не передается при наследовании, это конструкторы базового класса.
Таким образом, наследование реализует отношение **is-a** (является), объект класса Employee также является объектом класса Person:

```
1    static void Main(string[] args)
2    {
3    Person p = new Person { Name = "Tom"};
4    p.Display();
5    p = new Employee { Name = "Sam" };
6    p.Display();
7    Console.Read();
8        }
```

И поскольку объект Employee является также и объектом Person, то мы можем так определить переменную: Person p = new Employee().
По умолчанию все классы наследуются от базового класса **Object**, даже если мы явным образом не устанавливаем наследование. Поэтому выше определенные классы Person и Employee кроме своих собственных методов, также будут иметь и методы класса Object: ToString(), Equals(), GetHashCode() и GetType().
Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений:
Не поддерживается множественное наследование, класс может наследоваться только от одного класса.
При создании производного класса надо учитывать тип доступа к базовому классу - тип доступа к производному классу должен быть таким же, как и у базового класса, или

более строгим. То есть, если базовый класс у нас имеет тип доступа **internal**, то производный класс может иметь тип доступа **internal** или **private**, но не **public**.

Однако следует также учитывать, что если базовый и производный класс находятся в разных сборках (проектах), то в этом случае производый класс может наследовать только от класса, который имеет модификатор public.

Если класс объявлен с модификатором **sealed**, то от этого класса нельзя наследовать и создавать производные классы. Например, следующий класс не допускает создание наследников:

```
1   sealed class Admin
2   {
3   }
```

- Нельзя унаследовать класс от статического класса.

Доступ к членам базового класса из класса-наследника

Вернемся к нашим классам Person и Employee. Хотя Employee наследует весь функционал от класса Person, посмотрим, что будет в следующем случае:

```
1   class Employee : Person
2   {
3   public void Display()
4   {
5   Console.WriteLine(_name);
6       }
7    }
```

Этот код не сработает и выдаст ошибку, так как переменная _name объявлена с модификатором private и поэтому к ней доступ имеет только класс Person. Но зато в классе Person определено общедоступное свойство Name, которое мы можем использовать, поэтому следующий код у нас будет работать нормально:

```
1   class Employee : Person
2   {
3   public void Display()
4   {
5   Console.WriteLine(Name);
6       }
7    }
```

Таким образом, производный класс может иметь доступ только к тем членам базового класса, которые определены с модификаторами **private protected** (если базовый и производный класс находятся в одной сборке), **public**, **internal** (если базовый и производный класс находятся в одной сборке), **protected** и **protected internal**.

Ключевое слово base

Теперь добавим в наши классы конструкторы:

```
1   class Person
2   {
3   public string Name { get;  set; } 4
5   public Person(string name)
6   {
7   Name = name;
8       }
9
10  public void Display()
11  {
12  Console.WriteLine(Name);
13      }
```

```
14          }
15
    16  class Employee : Person
    17  {
    18  public string Company { get; set; } 19
    20  public Employee(string name, string company)
    21  : base(name)
22              {
23                  Company = company;
24              }
25          }
```

Класс Person имеет конструктор, который устанавливает свойство Name. Поскольку класс Employee наследует и устанавливает то же свойство Name, то логично было бы не писать по сто раз код установки, а как-то вызвать соответствующий код класса Person. К тому же свойств, которые надо установить в конструкторе базового класса, и параметров может быть гораздо больше.

С помощью ключевого слова **base** мы можем обратиться к базовому классу. В нашем случае в конструкторе класса Employee нам надо установить имя и компанию. Но имя мы передаем на установку в конструктор базового класса, то есть в конструктор класса Person, с помощью выражения base(name).

```
1   static void Main(string[] args)
2   {
3   Person p = new Person("Bill");
4   p.Display();
5   Employee emp = new Employee ("Tom", "Microsoft");
6   emp.Display();
7   Console.Read();
8       }
```

Конструкторы в производных классах

Конструкторы не передаются производному классу при наследовании. И если в базовом классе **не определен** конструктор по умолчанию без параметров, а только конструкторы с параметрами (как в случае с базовым классом Person), то в производном классе мы обязательно должны вызвать один из этих конструкторов через ключевое слово base. Например, из класса Employee уберем определение конструктора:

```
1   class Employee : Person
2   {
3   public string Company { get; set; }
4       }
```

В данном случае мы получим ошибку, так как класс Employee не соответствует классу Person, а именно не вызывает конструктор базового класса. Даже если бы мы добавили какой-нибудь конструктор, который бы устанавливал все те же свойства, то мы все равно бы получили ошибку:

```
1   public Employee(string name, string company)
2   {
3   Name = name;
4   Company = company;
5       }
```

То есть в классе Employee через ключевое слово **base** надо явным образом вызвать конструктор класса Person:

```
1    public Employee(string name, string company)
2           : base(name)
3    {
4       Company = company;
5    }
```

Либо в качестве альтернативы мы могли бы определить в базовом классе конструктор без параметров:

```
class Person
{
// остальной код класса
// конструктор по умолчанию
public Person()
{
FirstName = "Tom";
Console.WriteLine("Вызов конструктора без параметров");
        }
    }
```

Тогда в любом конструкторе производного класса, где нет обращения конструктору базового класса, все равно неявно вызывался бы этот конструктор по умолчанию. Например, следующий конструктор

```
public Employee(string company)
{
Company = company;
    }
```

Фактически был бы эквивалентен следующему конструктору:

```
public Employee(string company)
    :base()
 {
    Company = company;
 }
```

Порядок вызова конструкторов

При вызове конструктора класса сначала отрабатывают конструкторы базовых классов и только затем конструкторы производных. Например, возьмем следующие классы:

```
class Person
{
string name;
int age;

public Person(string name)
{
this.name = name;
Console.WriteLine("Person(string name)");
        }
public Person(string name, int age) : this(name)
{
this.age = age;
Console.WriteLine("Person(string name, int age)");
        }
    }
class Employee : Person
{
string company;

        public Employee(string name, int age, string company) : base(name, age)
```

```
22          {
23              this.company = company;
24              Console.WriteLine("Employee(string name, int age, string company)");
25          }
26      }
```

При создании объекта Employee:

```
1       Employee tom = new Employee("Tom", 22, "Microsoft");
```

Мы получим следующий консольный вывод:

```
Person(string name)
Person(string name, int age)
Employee(string name, int age, string company)
```

В итоге мы получаем следующую цепь выполнений.

1. Вначале вызывается конструктор Employee(string name, int age, string company). Он делегирует выполнение конструктору Person(string name, int age)
2. Вызывается конструктор Person(string name, int age), который сам пока не выполняется и передает выполнение конструктору Person(string name)
3. Вызывается конструктор Person(string name), который передает выполнение конструктору класса System.Object, так как это базовый по умолчанию класс для Person.
4. Выполняется конструктор System.Object.Object(), затем выполнение возвращается конструктору Person(string name)
5. Выполняется тело конструктора Person(string name), затем выполнение возвращается конструктору Person(string name, int age)
6. Выполняется тело конструктора Person(string name, int age), затем выполнение возвращается конструктору Employee(string name, int age, string company)
7. Выполняется тело конструктора Employee(string name, int age, string company). В итоге создается объект Employee

Практическая работа №9 Работа с объектами через интерфейсы

Интерфейс представляет ссылочный тип, который может определять некоторый функционал - набор методов и свойств без реализации. Затем этот функционал реализуют классы и структуры, которые применяют данные интерфейсы.

Определение интерфейса

Для определения интерфейса используется ключевое слово **interface**. Как правило, названия интерфейсов в C# начинаются с заглавной буквы **I**, например, IComparable, IEnumerable (так называемая венгерская нотация), однако это не обязательное требование, а больше стиль программирования.

Что может определять интерфейс? В целом интерфейсы могут определять следующие сущности:

Методы
Свойства
Индексаторы
События
Статические поля и константы (начиная с версии C# 8.0)

Однако интерфейсы не могут определять нестатические переменные.

Например, простейший интерфейс, который определяет все эти компоненты:

```
1    interface IMovable
2    {
3    // константа
4    const int minSpeed = 0;            // минимальная скорость
5    // статическая переменная
6    static int maxSpeed = 60;   // максимальная скорость
7    // метод
8    void Move();                       // движение
9    // свойство
10   string Name { get; set; }  // название 11
12   delegate void MoveHandler(string message);  // определение делегата для события
13   // событие
14   event MoveHandler MoveEvent;           // событие движения
15       }
```

В данном случае определен интерфейс IMovable, который представляет некоторый движущийся объект. Данный интерфейс содержит различные компоненты, которые описывают возможности движущегося объекта. То есть интерфейс описывает некоторый функционал, который должен быть у движущегося объекта.

Методы и свойства интерфейса могут не иметь реализации, в этом они сближаются с абстрактными методами и свойствами абстрактных классов. В данном случае интерфейс определяет метод Move, который будет представлять некоторое передвижение. Он не имеет реализации, не принимает никаких параметров и ничего не возвращает.

То же самое в данном случае касается свойства Name. На первый взгляд оно похоже на автоматическое свойство. Но в реальности это определение свойства в интерфейсе, которое не имеет реализации, а не автосвойство.

Еще один момент в объявлении интерфейса: если его члены - методы и свойства не имеют модификаторов доступа, но фактически по умолчанию доступ **public**, так как цель интерфейса - определение функционала для реализации его классом. Это касается также и констант и статических переменных, которые в классах и структурах по умолчанию имееют модификатор private. В интерфейсах же они имеют по умолчанию модификатор public. И например, мы могли бы обратиться к константе minSpeed и переменной maxSpeed интерфейса IMovable:

```
1    static void Main(string[] args)
```

```
    {
        Console.WriteLine(IMovable.maxSpeed);
        Console.WriteLine(IMovable.minSpeed);
    }
```

Но также, начиная с версии C# 8.0, мы можем явно указывать модификаторы доступа у компонентов интерфейса:

```
interface IMovable
{
public const int minSpeed = 0;          // минимальная скорость
private static int maxSpeed = 60;   // максимальная скорость
public void Move();
protected internal string Name { get; set; }          // название
public delegate void MoveHandler(string message);  // определение делегата для события
public event MoveHandler MoveEvent;          // событие движения
}
```

Начиная с версии C# 8.0 интерфейсы поддерживают реализацию методов и свойств по умолчанию. Это значит, что мы можем определить в интерфейсах полноценные методы и свойства, которые имеют реализацию как в обычных классах и структурах. Например, определим реализацию метода Move по умолчанию:

```
interface IMovable
{
// реализация метода по умолчанию
void Move()
{
Console.WriteLine("Walking");
    }
}
```

С реализацией свойств по умолчанию в интерфейсах дело обстоит несколько сложнее, поскольку мы не можем определять в интерфейсах нестатические переменные, соответственно в свойствах интерфейса мы не можем манипулировать состоянием объекта. Тем не менее реализацию по умолчанию для свойств мы тоже можем определять:

```
interface IMovable
{
void Move()
{
Console.WriteLine("Walking");
    }
// реализация свойства по умолчанию
// свойство только для чтения
int MaxSpeed { get { return 0; } }
  }
```

Стоит отметить, что если интерфейс имеет приватные методы и свойства (то есть с модификатором private), то они должны иметь реализацию по умолчанию. То же самое относится к любым статическим методам и свойствам (не обязательно приватным):

```
interface IMovable
{
public const int minSpeed = 0;          // минимальная скорость
private static int maxSpeed = 60;   // максимальная скорость
// находим время, за которое надо пройти расстояние distance со скоростью speed
static double GetTime(double distance, double speed) => distance / speed;
static int MaxSpeed
{
```

```
9               get { return maxSpeed; }
10              set
11              {
12                  if (value > 0) maxSpeed = value;
13              }
14          }
15      }
16  class Program
17  {
18  static void Main(string[] args)
19  {
20  Console.WriteLine(IMovable.MaxSpeed);
21  IMovable.MaxSpeed = 65;
22  Console.WriteLine(IMovable.MaxSpeed);
23  double time = IMovable.GetTime(100, 10);
24  Console.WriteLine(time);
25          }
26      }
```

Модификаторы доступа интерфейсов

Как и классы, интерфейсы по умолчанию имеют уровень доступа **internal**, то есть такой интерфейс доступен только в рамках текущего проекта. Но с помощью модификатора public мы можем сделать интерфейс общедоступным:

```
1   public interface IMovable
2   {
3   void Move();
4       }
```

Стоит отметить, что в Visual Studio есть специальный компонент для добавления нового интерфейса в отдельном файле. Для добавления интерфейса в проект можно нажать правой кнопкой мыши на проект и в появившемся контекстном меню выбрать **Add**-> **New Item...** и в диалоговом окне добавления нового компонента выбрать пункт **Interface**:

Применение интерфейсов

Интерфейс представляет некое описание типа, набор компонентов, который должен иметь тип данных. И, собственно, мы не можем создавать объекты интерфейса напрямую с помощью конструктора, как например, в классах:

```
1   IMovable m = new IMovable(); // ! Ошибка, так сделать нельзя
```

В конечном счете интерфейс предназначен для реализации в классах и структурах. Например, возьмем следующий интерфейс IMovable:

```
1   interface IMovable
2   {
3   void Move();
4       }
```

Затем какой-нибудь класс или структура могут применить данный интерфейс:

```
1       // применение интерфейса в классе
2       class Person : IMovable
3       {
4           public void Move()
5           {
6               Console.WriteLine("Человек идет");
7           }
8       }
9       // применение интерфейса в структуре
```

```
struct Car : IMovable
{
public void Move()
{
Console.WriteLine("Машина едет");
        }
    }
```

При применении интерфейса, как и при наследовании после имени класса или структуры указывается двоеточие и затем идут названия применяемых интерфейсов. При этом класс должен реализовать все методы и свойства применяемых интерфейсов, если эти методы и свойства не имеют реализации по умолчанию.

Если методы и свойства интерфейса не имеют модификатора доступа, то по умолчанию они являются публичными, при реализации этих методов и свойств в классе и структуре к ним можно применять только модификатор public.

Применение интерфейса в программе:

```
using System;

namespace HelloApp
{
interface IMovable
{
void Move();
        }
class Person : IMovable
{
public void Move()
{
Console.WriteLine("Человек идет");
            }
        }
struct Car : IMovable
{
public void Move()
{
Console.WriteLine("Машина едет");
            }
        }
class Program
{
static void Action(IMovable movable)
{
movable.Move();
            }
static void Main(string[] args)
{
Person person = new Person();
Car car = new Car();
Action(person);
Action(car);
Console.Read();
            }
        }
```

```
38      }
```

В данной программе определен метод Action(), который в качестве параметра принимает объект интерфейса IMovable. На момент написания кода мы можем не знать, что это будет за объект - какой-то класс или структура. Единственное, в чем мы можем быть уверены, что этот объект обязательно реализует метод Move и мы можем вызвать этот метод.

Иными словами, интерфейс - это контракт, что какой-то определенный тип обязательно реализует некоторый функционал.

Консольный вывод данной программы:

```
Человек идет
Машина едет
```

Реализация интерфейсов по умолчанию

Начиная с версии C# 8.0 интерфейсы поддерживают реализацию методов и свойств по умолчанию. Зачем это нужно? Допустим, у нас есть куча классов, которые реализуют некоторый интерфейс. Если мы добавим в этот интерфейс новый метод, то мы будем обязаны реализовать этот метод во всех классах, применяющих данный интерфейс. Иначе подобные классы просто не будут компилироваться. Теперь вместо реализации метода во всех классах нам достаточно определить его реализацию по умолчанию в интерфейсе. Если класс не реализует метод, будет применяться реализация по умолчанию.

```
1     class Program
2     {
3     static void Main(string[] args)
4     {
5     IMovable tom = new Person();
6     Car tesla = new Car();
7     tom.Move();               // Walking
8     tesla.Move();   // Driving
9           }
10        }
11
12    interface IMovable
13    {
14    void Move()
15    {
16    Console.WriteLine("Walking");
17          }
18        }
19    class Person : IMovable { }
20    class Car : IMovable
21    {
22    public void Move()
23    {
24    Console.WriteLine("Driving");
25          }
26        }
```

В данном случае интерфейс IMovable определяет реализацию по умолчанию для метода Move. Класс Person не реализует этот метод, поэтому он применяет реализацию по умолчанию в отличие от класса Car, который определяет свою реализацию для метода Move.

Стоит отметить, что хотя для объекта класса Person мы можем вызвать метод Move - ведь класс Person применяет интерфейс IMovable, тем не менее мы не можем написать так:

```
1     Person tom = new Person();
2     tom.Move();       // Ошибка - метод Move не определен в классе Person
```

Множественная реализация интерфейсов

Интерфейсы имеют еще одну важную функцию: в C# не поддерживается множественное наследование, то есть мы можем унаследовать класс только от одного класса, в отличие, скажем, от языка C++, где множественное наследование можно использовать. Интерфейсы позволяют частично обойти это ограничение, поскольку в C# класс может реализовать сразу несколько интерфейсов. Все реализуемые интерфейсы указываются через запятую:

```
myClass: myInterface1, myInterface2, myInterface3, ...
{

}
```

Рассмотрим на примере:

```
using System;

namespace HelloApp
{
    interface IAccount
    {
        int CurrentSum { get; }  // Текущая сумма на счету
        void Put(int sum);      // Положить деньги на счет
        void Withdraw(int sum); // Взять со счета
    }
    interface IClient
    {
        string Name { get; set; }
    }
    class Client : IAccount, IClient
    {
        int _sum; // Переменная для хранения суммы pub-
        lic string Name { get; set; }
        public Client(string name, int sum)
        {
            Name = name;
            _sum = sum;
        }

        public int CurrentSum { get { return _sum; } }

        public void Put(int sum) {  _sum += sum; } pub-

        lic void Withdraw(int sum)
        {
            if (_sum >= sum)
            {
                _sum -= sum;
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
```

```
            Client client = new Client("Tom", 200);
            client.Put(30);
            Console.WriteLine(client.CurrentSum); //230
            client.Withdraw(100);
            Console.WriteLine(client.CurrentSum); //130
            Console.Read();
        }
    }
}
```

В данном случае определены два интерфейса. Интерфейс IAccount определяет свойство CurrentSum для текущей суммы денег на счете и два метода Put и Withdraw для добавления денег на счет и изъятия денег. Интерфейс IClient определяет свойство для хранения имени клиента.

Обатите внимание, что свойства CurrentSum и Name в интерфейсах похожи на автосвойства, но это не автосвойства. При реализации мы можем развернуть их в полноценные свойства, либо же сделать автосвойствами.

Класс Client реализует оба интерфейса и затем применяется в программе.

Интерфейсы в преобразованиях типов

Все сказанное в отношении преобразования типов характерно и для интерфейсов. Поскольку класс Client реализует интерфейс IAccount, то переменная типа IAccount может хранить ссылку на объект типа Client:

```
// Все объекты Client являются объектами IAccount
IAccount account = new Client("Том", 200);
account.Put(200);
Console.WriteLine(account.CurrentSum); // 400
// Не все объекты IAccount являются объектами Client, необходимо явное приведение
Client client = (Client)account;
// Интерфейс IAccount не имеет свойства Name, необходимо явное приведение
string clientName = ((Client)account).Name;
```

Преобразование от класса к его интерфейсу, как и преобразование от производного типа к базовому, выполняется автоматически. Так как любой объект Client реализует интерфейс IAccount.

Обратное преобразование - от интерфейса к реализующему его классу будет аналогично преобразованию от базового класса к производному. Так как не каждый объект IAccount является объектом Client (ведь интерфейс IAccount могут реализовать и другие классы), то для подобного преобразования необходима операция приведения типов. И если мы хотим обратиться к методам класса Client, которые не определены в интерфейсе IAccount, но являются частью класса Client, то нам надо явным образом выполнить преобразование типов: string clientName = ((Client)account).Name;

сли класс применяет интерфейс, то этот класс должен реализовать все методы и свойства интерфейса, которые не имеют реализации по умолчанию. Однако также можно и не реализовать методы, сделав их абстрактными, переложив право их реализации на производные классы:

```
1   interface IMovable
2   {
3   void Move();
4       }
5   abstract class Person : IMovable
6   {
7   public abstract void Move();
8       }
9   class Driver : Person
10  {
11  public override void Move()
12  {
13  Console.WriteLine("Шофер ведет машину");
14          }
15      }
```

При реализации интерфейса учитываются также методы и свойства, унаследованные от базового класса. Например:

```
1   interface IAction
2   {
3   void Move();
4       }
5   class BaseAction
6   {
7   public void Move()
8   {
9   Console.WriteLine("Move in BaseAction");
10          }
11      }
12  class HeroAction : BaseAction, IAction
13  {
14  }
```

Здесь класс HeroAction реализует интерфейс IAction, однако для реализации метода Move из интерфейса применяется метод Move, унаследованный от базового класса BaseAction. Таким образом, класс HeroAction может не реализовать метод Move, так как этот метод уже определен в базовом классе BaseAction.

Следует отметить, что если класс одновременно наследует другой класс и реализует интерфейс, как в примере выше класс HeroAction, то название базового класса должно быть указано до реализуемых интерфейсов: class HeroAction : **BaseAction**, IAction

Изменение реализации интерфейсов в производных классах

Может сложиться ситуация, что базовый класс реализовал интерфейс, но в классе-наследнике необходимо изменить реализацию этого интерфейса. Что в этом случае делать? В этом случае мы можем использовать либо переопределение, либо сокрытие метода или свойства интерфейса.

Первый вариант - переопределение виртуальных/абстрактных методов:

```
1   interface IAction
2   {
```

```
    void Move();
}
class BaseAction : IAction
{
public virtual void Move()
{
Console.WriteLine("Move in BaseAction");
    }
}
class HeroAction : BaseAction
{
public override void Move()
{
Console.WriteLine("Move in HeroAction");
    }
}
```

В базовом классе BaseAction реализованный метод интерфейса определен как виртуальный (можно было бы также сделать его абстрактным), а в производном классе он переопределен.

При вызове метода через переменную интерфейса, если она ссылается на объект производного класса, будет использоваться реализация из производного класса:

```
BaseAction action1 = new HeroAction();
action1.Move();             // Move in HeroAction

IAction action2 = new HeroAction();
action2.Move();             // Move in HeroAction
```

Второй вариант - сокрытие метода в производном классе:

```
interface IAction
{
    void Move();
}
class BaseAction : IAction
{
    public void Move()
    {
        Console.WriteLine("Move in BaseAction");
    }
}
class HeroAction : BaseAction
{
    public new void Move()
    {
        Console.WriteLine("Move in HeroAction");
    }
}
```

Также используем эти классы:

```
BaseAction action1 = new HeroAction(); ac-
tion1.Move();             // Move in BaseAction

IAction action2 = new HeroAction();
action2.Move();             // Move in BaseAction
```

Так как интерфейс реализован именно в классе BaseAction, то через переменную action2 можно обратиться только к реализации метода Move из базового класса BaseAction.
Третий вариант - повторная реализация интерфейса в классе-наследнике:

```
interface IAction
{
void Move();
}
class BaseAction : IAction
{
public void Move()
{
Console.WriteLine("Move in BaseAction");
}
}
class HeroAction : BaseAction, IAction
{
public new void Move()
{
Console.WriteLine("Move in HeroAction");
}
}
```

В этом случае реализации этого метода из базового класса будет игнорироваться:

```
BaseAction action1 = new HeroAction();
action1.Move();             // Move in BaseAction

IAction action2 = new HeroAction();
action2.Move();             // Move in HeroAction

HeroAction action3 = new HeroAction();
action3.Move();             // Move in HeroAction
```

Также стоит отметить, что в случае с переменной action1 по-прежнему действует ранее связывание, в силу которого через эту переменную можно вызвать реализацию метода Move только из базового класса, который эта переменная представляет.
Четвертый вариант: явная реализация интерфейса:

```
interface IAction
{
void Move();
}
class BaseAction : IAction
{
public void Move()
{
Console.WriteLine("Move in BaseAction");
}
}
class HeroAction : BaseAction, IAction
{
public new void Move()
{
Console.WriteLine("Move in HeroAction");
}
void IAction.Move()
```

```
19  {
20          Console.WriteLine("Move in IAction");
21      }
22  }
```

В этом случае для переменной IAction будет использоваться явная реализация интерфейса IAction, а для переменной HeroAction по прежнему будет использоваться неявная реализация:

```
1  BaseAction action1 = new HeroAction();
2  action1.Move();          // Move in BaseAction
3
4  IAction action2 = new HeroAction();
5  action2.Move();          // Move in IAction
6
7  HeroAction action3 = new HeroAction();
8  action3.Move();          // Move in HeroAction
```

Практическая работа №11 Работа с типом данных структура

Наряду с классами структуры представляют еще один способ создания собственных типов данных в C#. Более того многие примитивные типы, например, int, double и т.д., по сути являются структурами.

Например, определим структуру, которая представляет человека:

```
1    struct User
2    {
3    public string name;
4    public int age; 5
6    public void DisplayInfo()
7    {
8    Console.WriteLine($"Name: {name}  Age: {age}");
9        }
10   }
```

Как и классы, структуры могут хранить состояние в виде переменных и определять поведение в виде методов. Так, в данном случае определены две переменные - name и age для хранения соответственно имени и возраста человека и метод DisplayInfo для вывода информации о человеке.

Используем эту структуру в программе:

```
1    using System;
2
3    namespace HelloApp
4    {
5        struct User
6        {
7            public string name;
8            public int age;
9
10           public void DisplayInfo()
11           {
12               Console.WriteLine($"Name: {name}  Age: {age}");
13           }
14       }
15
16       class Program
17       {
18           static void Main(string[] args)
19           {
20               User tom;
21               tom.name = "Tom";
22               tom.age = 34;
23               tom.DisplayInfo();
24
25               Console.ReadKey();
26           }
27       }
28   }
```

В данном случае создается объект tom. У него устанавливаются значения глобальных переменных, и затем выводится информация о нем.

Конструкторы структуры

Как и класс, структура может определять конструкторы. Но в отличие от класса нам не обязательно вызывать конструктор для создания объекта структуры:

```
User tom;
```

Однако если мы таким образом создаем объект структуры, то обязательно надо проинициализировать все поля (глобальные переменные) структуры перед получением их значений или перед вызовом методов структуры. То есть, например, в следующем случае мы получим ошибку, так как обращение к полям и методам происходит до присвоения им начальных значений:

```
User tom;
int x = tom.age;      // Ошибка
tom.DisplayInfo();  // Ошибка
```

Также мы можем использовать для создания структуры конструктор без параметров, который есть в структуре по умолчанию и при вызове которого полям структуры будет присвоено значение по умолчанию (например, для числовых типов это число 0):

```
User tom = new User();
tom.DisplayInfo();  // Name:   Age: 0
```

Обратите внимание, что при использовании конструктора по умолчанию нам не надо явным образом инициализировать поля структуры.

Также мы можем определить свои конструкторы. Например, изменим структуру User:

```
using System;

namespace HelloApp
{
    struct User
    {
        public string name;
        public int age;
        public User(string name, int age)
        {
            this.name = name;
            this.age = age;
        }
        public void DisplayInfo()
        {
            Console.WriteLine($"Name: {name}  Age: {age}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            User tom = new User("Tom", 34);
            tom.DisplayInfo();

            User bob = new User();
            bob.DisplayInfo();

            Console.ReadKey();
        }
    }
```

```
33          }
```

Важно учитывать, что если мы определяем конструктор в структуре, то он должен инициализировать все поля структуры, как в данном случае устанавливаются значения для переменных name и age.

Также, как и для класса, можно использовать инициализатор для создания структуры:

```
1     User person = new User { name = "Sam", age = 31 };
```

Но в отличие от класса нельзя инициализировать поля структуры напрямую при их объявлении, например, следующим образом:

```
1     struct User
2     {
3         public string name = "Sam";     // ! Ошибка
4         public int age = 23;            // ! Ошибка
5         public void DisplayInfo()
6         {
7             Console.WriteLine($"Name: {name}  Age: {age}");
8         }
9     }
```

Практическая работа №12 Коллекции. Параметризованные классы

Аннотация: Списки, очереди, двоичные массивы, хэш-таблицы, словари – все это коллекции. Существуют различные типы (классы) коллекций. Объект – представитель данного класса коллекции характеризуется множеством функциональных признаков, определяющих способы работы с элементами (неважно какого типа), которые образуют данную коллекцию

Ключевые слова: пространство имен, IList, ICollection, BinarySearch, LastIndexOf, открытый метод, dequeue, enqueue, peek, интерфейс, вызов метода, значение, перечисление

Обзор

Пространство имен System.Collections содержит классы и интерфейсы, которые определяют различные коллекции объектов.

Классы

Класс	**Описание**
ArrayList	Служит для реализации интерфейса *IList* с помощью массива с динамическим изменением размера по требованию
BitArray	Управляет компактным массивом двоичных значений, представленных логическими величинами, где значение true соответствует 1, а значение false соответствует 0
CaseInsensitiveComparer	Проверяет равенство двух объектов без учета регистра строк
CaseInsensitiveHashCodeProvider	Предоставляет хэш-код объекта, используя алгоритм хэширования, при котором не учитывается регистр строк
CollectionBase	Предоставляет абстрактный (MustInherit в Visual Basic) базовый класс для коллекции со строгим типом
Comparer	Проверяет равенство двух объектов с учетом регистра строк
DictionaryBase	Предоставляет абстрактный (MustInherit в Visual Basic) базовый класс для коллекции пар "ключ-значение" со строгим типом
Hashtable	Предоставляет коллекцию пар "ключ-значение", которые упорядочены по хэш-коду ключа
Queue	Предоставляет коллекцию объектов, которая обслуживается по принципу "первым пришел — первым вышел" (FIFO)
ReadOnlyCollectionBase	Предоставляет абстрактный (MustInherit в Visual Basic) базовый класс для коллекции со строгим типом, которая доступна только для чтения
SortedList	Предоставляет коллекцию пар "ключ-значение", которые упорядочены по ключам. Доступ к парам можно получить по ключу и по индексу
Stack	Представляет коллекцию объектов, которая обслуживается по принципу "последним пришел — первым вышел" (LIFO)

Интерфейсы

Интерфейс	**Описание**
ICollection	Определяет размер, перечислители и методы синхронизации для всех коллекций
IComparer	Предоставляет другим приложениям метод для сравнения

	двух объектов
IDictionary	Предоставляет коллекцию пар "ключ-значение"
IDictionaryEnumerator	Осуществляет нумерацию элементов словаря
IEnumerable	Предоставляет перечислитель, который поддерживает простое перемещение по коллекции
IEnumerator	Поддерживает простое перемещение по коллекции
IHashCodeProvider	Предоставляет хэш-код объекта, используя пользовательскую хэш-функцию
IList	Предоставляет коллекцию объектов, к которым можно получить доступ отдельно, по индексу

Структуры

Структура	**Описание**
DictionaryEntry	Определяет в словаре пару "ключ-значение", которая может быть задана или получена

Примеры

3. ArrayList

Неполный перечень свойств и методов приводится ниже.

Конструктор

ArrayList	Перегружен. Инициализирует новый экземпляр класса ArrayList

Свойства

Capacity	Возвращает или задает число элементов, которое может содержать класс ArrayList
Count	Возвращает число элементов, которое в действительности хранится в классе ArrayList
IsFixedSize	Возвращает значение, показывающее, имеет ли класс ArrayList фиксированный размер
IsReadOnly	Возвращает значение, определяющее, доступен ли класс ArrayList только для чтения
IsSynchronized	Возвращает значение, определяющее, является ли доступ к классу ArrayList синхронизированным (потокобезопасным)
Item	Возвращает или задает элемент с указанным индексом. В C# это свойство является индексатором класса ArrayList

Методы

Add	Добавляет объект в конец класса ArrayList
AddRange	Добавляет элементы интерфейса *ICollection* в конец класса ArrayList
BinarySearch	Перегружен. Использует алгоритм двоичного поиска для нахождения определенного элемента в отсортированном классе ArrayList или в его части
Clear	Удаляет все элементы из класса ArrayList
Clone	Создает неполную копию класса ArrayList
Contains	Определяет, принадлежит ли элемент классу ArrayList
CopyTo	Перегружен. Копирует класс ArrayList или его часть в одномерный массив
FixedSize	Перегружен. Возвращает обертку списка фиксированного размера, в которой элементы можно изменять, но нельзя добавлять или удалять
GetEnumerator	Перегружен. Возвращает перечислитель, который может осуществлять просмотр всех элементов класса ArrayList
GetRange	Возвращает класс ArrayList, который представляет собой подмножество элементов в исходном классе ArrayList
IndexOf	Перегружен. Возвращает отсчитываемый от нуля индекс первого найденного элемента в классе ArrayList или в его части

Insert	Вставляет элемент в класс ArrayList по указанному индексу
InsertRange	Вставляет элементы коллекции в класс ArrayList по указанному индексу
LastIndexOf	Перегружен. Возвращает отсчитываемый от нуля индекс последнего найденного элемента в классе ArrayList или в его части
Remove	Удаляет первый найденный объект из класса ArrayList
RemoveAt	Удаляет элемент с указанным индексом из класса ArrayList
RemoveRange	Удаляет диапазон элементов из класса ArrayList
Repeat	Возвращает класс ArrayList, элементы которого являются копиями указанного значения
Reverse	Перегружен. Изменяет порядок элементов в классе ArrayList или в его части на обратный
SetRange	Копирует элементы коллекции в диапазон элементов класса ArrayList
Sort	Перегружен. Сортирует элементы в классе ArrayList или в его части
ToArray	Перегружен. Копирует элементы класса ArrayList в новый массив
TrimToSize	Задает значение емкости, равное действительному количеству элементов в классе ArrayList

```
using System;
    using          System.Collections;
    public class SamplesArrayList  {

    public static void Main()  {

    // Создается и инициализируется объект ArrayList.
    ArrayList      myAL      =      new      ArrayList();
    myAL.Add("Россия,");
    myAL.Add("вперед");
    myAL.Add("!");

    // Свойства и значения ArrayList.
    Console.WriteLine( "myAL" );
    Console.WriteLine( "\tCount:   {0}",   myAL.Count   );
    Console.WriteLine(  "\tCapacity:  {0}",  myAL.Capacity
    ); Console.Write( "\tValues:" );
    PrintValues( myAL );
    }

    public static void PrintValues( IEnumerable myList )
    {
    // Для эффективной работы с объектом ArrayList
    // создается перечислитель...
    //      Перечислитель      обеспечивает      перебор      элементов.
    System.Collections.IEnumerator myEnumerator
          = myList.GetEnumerator(); // Перечислитель для myList
    while (myEnumerator.MoveNext())
     Console.Write( "\t{0}", myEnumerator.Current );
    }
    }
```

Листинг 12.1.

Результат:

```
myAL
       Count:    3
       Capacity: 16
```

Values: Россия , вперед !

4. BitArray

Неполный перечень свойств и методов приводится ниже.

Конструкторы

BitArray	Перегружен. Инициализирует новый экземпляр класса BitArray, для которого могут быть указаны емкость и начальные значения

Открытые свойства

Count	Возвращает число элементов, которое хранится в классе BitArray
IsReadOnly	Возвращает значение, определяющее, доступен ли класс BitArray только для чтения
IsSynchronized	Возвращает значение, определяющее, является ли доступ к классу BitArray синхронизированным (потокобезопасным)
Item	Возвращает или задает значение бита по указанному адресу в классе BitArray В языке C# это свойство является индексатором класса BitArray
Length	Возвращает или задает число элементов в классе BitArray
SyncRoot	Возвращает объект, который может быть использован для синхронизации доступа к классу BitArray

Открытые методы

And	Выполняет поразрядную операцию логического умножения элементов текущего класса BitArray с соответствующими элементами указанного класса BitArray
Clone	Создает неполную копию класса BitArray
CopyTo	Копирует целый класс BitArray в совместимый одномерный массив класса Array, начиная с указанного индекса конечного массива
Get	Возвращает значение бита по указанному адресу в классе BitArray
GetEnumerator	Возвращает перечислитель, который может осуществлять просмотр всех элементов класса BitArray
Not	Преобразовывает все двоичные значения в текущем классе BitArray таким образом, чтобы каждому элементу со значением true было присвоено значение false, а каждому элементу со значением false было присвоено значение true
Or	Выполняет поразрядную операцию логического сложения элементов текущего класса BitArray с соответствующими элементами указанного класса BitArray
Set	Задает указанное значение биту по указанному адресу в классе BitArray
SetAll	Задает определенное значение всем битам в классе BitArray
Xor	Выполняет поразрядную операцию "исключающее ИЛИ" для элементов текущего класса BitArray и соответствующих элементов указанного класса BitArray

Пример использования:

```
using System;
    using        System.Collections;
    public class SamplesBitArray  {

      public static void Main()  {

        // Creates and initializes several BitArrays.
        BitArray myBA1 = new BitArray( 5 );

        BitArray myBA2 = new BitArray( 5, false );
```

```
    byte[] myBytes = new byte[5] { 1, 2, 3, 4, 5 };
    BitArray myBA3 = new BitArray( myBytes );

    bool[] myBools = new bool[5] { true, false, true, true, false };
    BitArray myBA4 = new BitArray( myBools );

    int[] myInts = new int[5] { 6, 7, 8, 9, 10 };
    BitArray myBA5 = new BitArray( myInts );

    // Displays the properties and values of the BitArrays.
    Console.WriteLine( "myBA1" );
    Console.WriteLine( "\tCount:    {0}", myBA1.Count );
    Console.WriteLine( "\tLength:   {0}", myBA1.Length );
    Console.WriteLine( "\tValues:" );
    PrintValues( myBA1, 8 );

    Console.WriteLine( "myBA2" );
    Console.WriteLine( "\tCount:    {0}", myBA2.Count );
    Console.WriteLine( "\tLength:   {0}", myBA2.Length );
    Console.WriteLine( "\tValues:" );
    PrintValues( myBA2, 8 );

    Console.WriteLine( "myBA3" );
    Console.WriteLine( "\tCount:    {0}", myBA3.Count );
    Console.WriteLine( "\tLength:   {0}", myBA3.Length );
    Console.WriteLine( "\tValues:" );
    PrintValues( myBA3, 8 );

    Console.WriteLine( "myBA4" );
    Console.WriteLine( "\tCount:    {0}", myBA4.Count );
    Console.WriteLine( "\tLength:   {0}", myBA4.Length );
    Console.WriteLine( "\tValues:" );
    PrintValues( myBA4, 8 );
    Console.WriteLine( "myBA5" );
    Console.WriteLine( "\tCount:    {0}", myBA5.Count );
    Console.WriteLine( "\tLength:   {0}", myBA5.Length );
    Console.WriteLine( "\tValues:" );
    PrintValues( myBA5, 8 );
  }


  public static void PrintValues( IEnumerable myList, int myWidth )
  {
  System.Collections.IEnumerator myEnumerator = myList.GetEnumerator();
  int i = myWidth;
  while ( myEnumerator.MoveNext() )
 {
  if ( i <= 0 )
  {
   i = myWidth;
   Console.WriteLine();
  }
```

```
            i--;
            Console.Write( "\t{0}", myEnumerator.Current );
        }
          Console.WriteLine();
        }
    }
```

Листинг 12.2.

Результат:

```
myBA1
        Count:    5
        Length:        5
        Values:
        False    False    False    False    False
    myBA2
        Count:    5
        Length:        5
        Values:
        False    False    False    False    False
    myBA3
        Count:    40
        Length:       40
        Values:
        True   False      False      False      False      False      False      False
        False  True       False      False      False      False      False      False
        True   True       False      False      False      False      False      False
        False  False      True       False      False      False      False      False
        True   False      True       False      False      False      False      False
    myBA4
        Count:    5
        Length:        5
        Values:
        True    False    True    True    False
    myBA5
        Count:    160
        Length:     160
        Values:
        False  True       True       False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        True   True       True       False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      True       False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        True   False      False      True       False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
        False  False      False      False      False      False      False      False
```

False	True	False	True	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False

Листинг 12.3.

5. Queue

Неполный перечень свойств и методов.

Конструктор

Queue	Перегружен. Инициализирует новый экземпляр класса Queue

Открытые свойства

Count	Возвращает число элементов, которое хранится в классе Queue
SyncRoot	Получает объект, который может быть использован для синхронизации доступа к классу Queue

Открытые методы

Clear	Удаляет все объекты из класса Queue
Clone	Создает поверхностную копию класса Queue
Contains	Определяет, принадлежит ли элемент классу Queue
CopyTo	Копирует элементы класса Queue в существующий одномерный массив класса Array по указанному адресу
Dequeue	Удаляет и возвращает объект в начале класса Queue
Enqueue	Добавляет объект в конец класса Queue
GetEnumerator	Возвращает перечислитель, который может перемещаться по классу Queue
Peek	Возвращает объект в начале класса Queue, но не удаляет его
ToArray	Копирует элементы класса Queue в новый массив
TrimToSize	Задает значение емкости, равное действительному количеству элементов в классе Queue

Пример:

```
using System;
    using      System.Collections;
    public class SamplesQueue
    {

      public static void Main()
      {
        // Creates and initializes a new Queue.
        Queue myQ = new Queue();
        myQ.Enqueue("Россия,");
        myQ.Enqueue("вперед");
        myQ.Enqueue("!");

      // Displays the properties and values of the Queue.
       Console.WriteLine( "myQ" );
       Console.WriteLine( "\tCount:    {0}", myQ.Count );
       Console.Write( "\tValues:" );
       PrintValues( myQ );
      }


    public static void PrintValues( IEnumerable myCollection )
     {
```

```
        System.Collections.IEnumerator myEnumerator
                                = myCollection.GetEnumerator();
          while ( myEnumerator.MoveNext() )
            Console.Write( "\t{0}", myEnumerator.Current );
        }
    }
```

Листинг 12.4.

Результат:

```
myQ
        Count:    3
        Values:    Россия ,   вперед    !
```

6. Stack

Открытые конструкторы	
Stack	Перегружен. Инициализирует новый экземпляр класса Stack
Открытые свойства	
Count	Возвращает число элементов, которое хранится в классе Stack
IsSynchronized	Возвращает значение, определяющее, является ли доступ к классу Stack синхронизированным (потокобезопасным)
SyncRoot	Возвращает объект, который может быть использован для синхронизации доступа к классу Stack
Открытые методы	
Clear	Удаляет все объекты из класса Stack
Clone	Создает неполную копию класса Stack
Contains	Определяет, принадлежит ли элемент классу Stack
CopyTo	Копирует элементы класса Stack в существующий одномерный массив класса Array, начиная с указанного индекса массива
GetEnumerator	Интерфейс IEnumerator для класса Stack
Peek	Возвращает самый верхний объект класса Stack, но не удаляет его
Pop	Удаляет и возвращает верхний объект класса Stack
Push	Вставляет объект в начало класса Stack
Synchronized	Возвращает синхронизированную (потокобезопасную) обертку класса Stack
ToArray	Копирует элементы класса Stack в новый массив

Класс Stack допускает в качестве действительного значение "пустая ссылка", а также допускает наличие повторяющихся элементов.

Ниже приведен пример создания и инициализации класса Stack и способ вывода его значений:

```
using System;
    using      System.Collections;
    public class SamplesStack  {

     public static void Main()
     {

     // Creates and initializes a new Stack.
         Stack   myStack   =   new   Stack();
         myStack.Push("Hello");
         myStack.Push("world");
         myStack.Push("!");
```

```
        // Displays the properties and values of the Stack.
        Console.WriteLine( "myStack" );
        Console.WriteLine( "\tCount:    {0}", myStack.Count );
        Console.Write( "\tValues:" );
        PrintValues( myStack );
      }


    public static void PrintValues( IEnumerable myCollection )
     {
     System.Collections.IEnumerator myEnumerator = myCollection.GetEnumerator();
      while ( myEnumerator.MoveNext() )
      Console.Write( "\t{0}", myEnumerator.Current );
      Console.WriteLine();
     }
   }
```

Листинг 12.5.

Результат:

```
myStack
      Count:    3
      Values:    !    world    Hello
```

Перечислитель

Наследует *интерфейс* IEnumerator, который является основным для всех перечислителей. Поддерживает простое перемещение по коллекции.

Открытые свойства

Current	Возвращает текущий элемент коллекции

Открытые методы

MoveNext	Перемещает перечислитель на следующий элемент коллекции
Reset	Устанавливает перечислитель в исходное положение перед первым элементом коллекции

Перечислитель позволяет считывать (только считывать!) информацию (данные) из коллекции. Перечислители не используются для изменения содержания коллекции. Для этого применяются специфические методы данной коллекции (*Enqueue*, *Dequeue*, Push, Pop).

Вновь созданный перечислитель размещается перед первым элементом коллекции. Метод Reset возвращает перечислитель обратно в положение перед первым элементом коллекции.

В этом положении обращение к свойству Current приводит к возникновению исключения. Поэтому необходимо вызвать метод MoveNext, чтобы переместить перечислитель на первый элемент коллекции до считывания значения свойства Current.

Свойство Current не меняет своего значения (возвращает ссылку на один и тот же член коллекции), пока не будет вызван метод MoveNext или Reset.

Метод MoveNext обеспечивает изменение значения свойства Current.

Завершение перемещения по коллекции приводит к установке перечислителя после последнего элемента коллекции. При этом *вызов метода* MoveNext возвращает *значение* false. Если последний *вызов метода* MoveNext вернул *значение* false, обращение к свойству Current приводит к возникновению исключения. Последовательный вызов методов Reset и MoveNext приводит к перемещению перечислителя на первый элемент коллекции.

Перечислитель действителен до тех пор, пока в коллекцию не вносятся изменения. Если в коллекцию вносятся изменения (добавляются или удаляются элементы коллекции), перечислитель становится недействительным, а следующий вызов методов MoveNext или Reset приводит к возникновению исключения InvalidOperationException.
После изменения коллекции в промежутке между вызовом метода MoveNext и новым обращением к свойству Current, свойство Current возвращает текущий элемент коллекции, даже если перечислитель уже недействителен.
Перечислитель не имеет монопольного доступа к коллекции, поэтому *перечисление* в коллекции не является потокобезопасной операцией. Даже при синхронизации коллекции другие потоки могут изменить ее, что приводит к созданию исключения при перечислении. Чтобы обеспечить потокобезопасность при перечислении, можно либо заблокировать коллекцию на все время перечисления, либо перехватывать исключения, которые возникают в результате изменений, внесенных другими потоками.

Практическая работа №13 Использование регулярных выражений

Синтаксис регулярных выражений

Рассмотрим вкратце некоторые элементы синтаксиса регулярных выражений:

- ^: соответствие должно начинаться в начале строки (например, выражение @"^пр\w*" соответствует слову "привет" в строке "привет мир")
- **$**: конец строки (например, выражение @"\w*ир$" соответствует слову "мир" в строке "привет мир", так как часть "ир" находится в самом конце)
- **.**: знак точки определяет любой одиночный символ (например, выражение "м.р" соответствует слову "мир" или "мор")
- ***: предыдущий символ повторяется 0 и более раз
- +: предыдущий символ повторяется 1 и более раз
- **?**: предыдущий символ повторяется 0 или 1 раз
- **\s**: соответствует любому пробельному символу
- **\S**: соответствует любому символу, не являющемуся пробелом
- **\w**: соответствует любому алфавитно-цифровому символу
- **\W**: соответствует любому не алфавитно-цифровому символу
- **\d**: соответствует любой десятичной цифре
- **\D** : соответствует любому символу, не являющемуся десятичной цифрой

Это только небольшая часть элементов. Более подробное описание синтаксиса регулярных выражений можно найти на msdn в статье Элементы языка регулярных выражений — краткий справочник.

Теперь посмотрим на некоторые примеры использования. Возьмем первый пример с скороговоркой "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа" и найдем в ней все слова, где встречается корень "губ":

```
1  string s = "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа";
2  Regex regex = new Regex(@"\w*губ\w*");
```

Так как выражение \w* соответствует любой последовательности алфавитно-цифровых символов любой длины, то данное выражение найдет все слова, содержащие корень "губ".

Второй простенький пример - нахождение телефонного номера в формате 111-111-1111:

```
1  string s = "456-435-2318";
2  Regex regex = new Regex(@"\d{3}-\d{3}-\d{4}");
```

Если мы точно знаем, сколько определенных символов должно быть, то мы можем явным образом указать их количество в фигурных скобках: \d{3} - то есть в данном случае три цифры.

Мы можем не только задать поиск по определенным типам символов - пробелы, цифры, но и задать конкретные символы, которые должны входить в регулярное выражение. Например, перепишем пример с номером телефона и явно укажем, какие символы там должны быть:

```
1  string s = "456-435-2318";
2  Regex regex = new Regex("[0-9]{3}-[0-9]{3}-[0-9]{4}");
```

В квадратных скобках задается диапазон символов, которые должны в данном месте встречаться. В итоге данный и предыдущий шаблоны телефонного номера будут эквивалентны.

Также можно задать диапазон для алфавитных символов: Regex regex = new Regex("[a- v]{5}"); - данное выражение будет соответствовать любому сочетанию пяти символов, в котором все символы находятся в диапазоне от a до v.

Можно также указать отдельные значения: Regex regex = new Regex(@"[2]*-[0-9]{3}-\d{4}");. Это выражение будет соответствовать, например, такому номеру телефона "222-222-2222" (так как первые числа двойки)

С помощью операции | можно задать альтернативные символы: Regex regex = new Regex(@"[2\|3]{3}-[0-9]{3}-\d{4}");. То есть первые три цифры могут содержать только двойки или тройки. Такой шаблон будет соответствовать, например, строкам "222-222-

2222" и "323-435-2318". А вот строка "235-435-2318" уже не подпадает под шаблон, так как одной из трех первых цифр является цифра 5.

Итак, у нас такие символы, как *, + и ряд других используются в качестве специальных символов. И возникает вопрос, а что делать, если нам надо найти, строки, где содержится точка, звездочка или какой-то другой специальный символ? В этом случае нам надо просто экранировать эти символы слешем:

```
Regex regex = new Regex(@"[2\|3]{3}\.[0-9]{3}\.\d{4}");
// этому выражению будет соответствовать строка "222.222.2222"
```

Проверка на соответствие строки формату

Нередко возникает задача проверить корректность данных, введенных пользователем. Это может быть проверка электронного адреса, номера телефона, Класс Regex предоставляет статический метод **IsMatch**, который позволяет проверить входную строку с шаблоном на соответствие:

```
string pattern = @"^(?("")(""[^""]+?""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\*\+/=\?\^`\{\}\|~\w])*)(?<=[
        @"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]*[0-9a-z]*\.)+[a-z0-9]{2,17}))$";
while (true)
{
Console.WriteLine("Введите адрес электронной почты");
string email = Console.ReadLine(); 7
if (Regex.IsMatch(email, pattern, RegexOptions.IgnoreCase))
{
Console.WriteLine("Email подтвержден");
break;
    }
else
{
Console.WriteLine("Некорректный email");
    }
  }
```

Переменная pattern задает регулярное выражение для проверки адреса электронной почты. Данное выражение предлагает нам Microsoft на страницах msdn.

Для проверки соответствия строки шаблону используется метод IsMatch: Regex.IsMatch(email, pattern, RegexOptions.IgnoreCase). Последний параметр указывает, что регистр можно игнорировать. И если введенная строка соответствует шаблону, то метод возвращает true.

Замена и метод Replace

Класс Regex имеет метод Replace, который позволяет заменить строку, соответствующую регулярному выражению, другой строкой:

```
string s = "Мама  мыла  раму. ";
string pattern = @"\s+";
string target = " ";
Regex regex = new Regex(pattern);
string result = regex.Replace(s, target);
```

Данная версия метода Replace принимает два параметра: строку с текстом, где надо выполнить замену, и сама строка замены. Так как в качестве шаблона выбрано выражение "\s+ (то есть наличие одного и более пробелов), метод Replace проходит по всему тексту и заменяет несколько подряд идущих пробелов ординарными.

Практическая работа №14 Операции со списками

Класс List<T> из пространства имен System.Collections.Generic представляет простейший список однотипных объектов.

Среди его методов можно выделить следующие:

void Add(T item): добавление нового элемента в список

void AddRange(ICollection collection): добавление в список коллекции или массива

int BinarySearch(T item): бинарный поиск элемента в списке. Если элемент найден, то метод возвращает индекс этого элемента в коллекции. При этом список должен быть отсортирован.

int IndexOf(T item): возвращает индекс первого вхождения элемента в списке

void Insert(int index, T item): вставляет элемент item в списке на позицию index

bool Remove(T item): удаляет элемент item из списка, и если удаление прошло успешно, то возвращает true

void RemoveAt(int index): удаление элемента по указанному индексу index

void Sort(): сортировка списка Посмотрим реализацию списка на примере:

```
1   using System;
2   using System.Collections.Generic;
3
4   namespace Collections
5   {
6   class Program
7   {
8   static void Main(string[] args)
9   {
10  List<int> numbers = new List<int>() { 1, 2, 3, 45 };
11  numbers.Add(6); // добавление элемента 12
13              numbers.AddRange(new int[] { 7, 8, 9 });
14
15              numbers.Insert(0, 666); // вставляем на первое место в списке число 666
16
17              numbers.RemoveAt(1); //  удаляем второй элемент
18
19  foreach (int i in numbers)
20  {
21  Console.WriteLine(i);
22              }
23
24  List<Person> people = new List<Person>(3);
25  people.Add(new Person() { Name = "Том" });
26  people.Add(new Person() { Name = "Билл" }); 27
28  foreach (Person p in people)
29  {
30  Console.WriteLine(p.Name);
31              }
32
33              Console.ReadLine();
34          }
35      }
36
37  class Person
38  {
39  public string Name { get; set; }
```

```
40              }
41          }
```

Здесь у нас создаются два списка: один для объектов типа int, а другой - для объектов Person. В первом случае мы выполняем начальную инициализацию списка: List<int> numbers = new List<int>() { 1, 2, 3, 45 };

Во втором случае мы используем другой конструктор, в который передаем начальную емкость списка: List<Person> people = new List<Person>(3);. Указание начальной емкости списка (capacity) позволяет в будущем увеличить производительность и уменьшить издержки на выделение памяти при добавлении элементов. Также начальную емкость можно установить с помощью свойства Capacity, которое имеется у класса List.

можем выделить несколько основных отношений: наследование, реализация, ассоциация, композиция и агрегация.

Наследование

Наследование является базовым принципом ООП и позволяет одному классу (наследнику) унаследовать функционал другого класса (родительского). Нередко отношения наследования еще называют генерализацией или обобщением. Наследование определяет отношение **IS A**, то есть "является". Например:

```
1    class User
2    {
3    public int Id { get; set; }
4    public string Name { get; set; }
5        }
6
7    class Manager : User
8    {
9    public string Company{ get; set; }
10       }
```

В данном случае используется наследование, а объекты класса Manager также **являются** и объектами класса User.

С помощью диаграмм UML отношение между классами выражается в незакрашенной стрелочке от класса-наследника к классу-родителю:

Реализация

Реализация предполагает определение интерфейса и его реализация в классах. Например, имеется интерфейс IMovable с методом Move, который реализуется в классе Car:

```
1    public interface IMovable
2    {
3    void Move();
4        }
5    public class Car : IMovable
6    {
7    public void Move()
8    {
9    Console.WriteLine("Машина едет");
10           }
11       }
```

С помощью диаграмм UML отношение реализации также выражается в незакрашенной стрелочке от класса к интерфейсу, только линия теперь пунктирная:

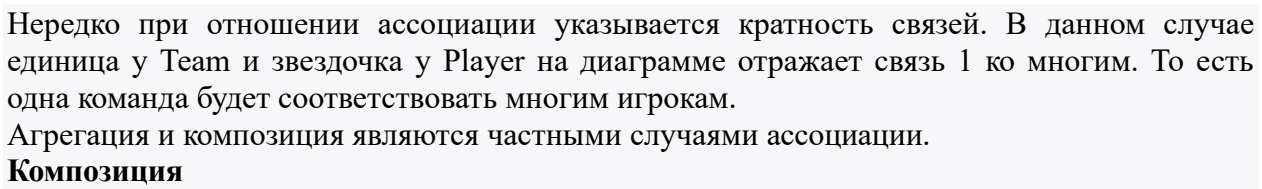
Ассоциация

Ассоциация - это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа. Например, объект одного типа содержит или использует объект другого типа. Например, игрок играет в определенной команде:

```
1    class Team
2    {
3
4        }
5    class Player
6    {
7    public Team Team { get; set; }
```

```
8       }
```

Класс Player связан отношением ассоциации с класом Team. На схемах UML ассоциация обозначается в виде обычно стрелки:



Нередко при отношении ассоциации указывается кратность связей. В данном случае единица у Team и звездочка у Player на диаграмме отражает связь 1 ко многим. То есть одна команда будет соответствовать многим игрокам.

Агрегация и композиция являются частными случаями ассоциации.

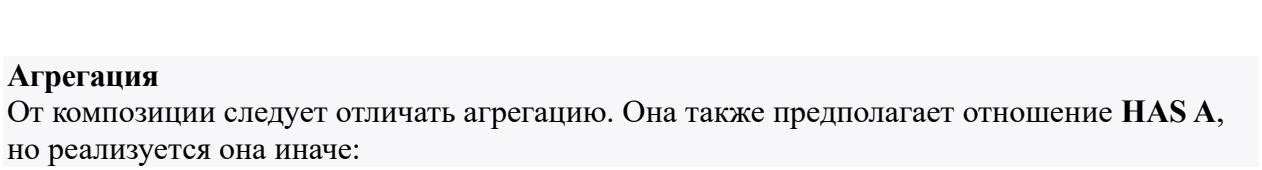
Композиция

Композиция определяет отношение **HAS A**, то есть отношение "имеет". Например, в класс автомобиля содержит объект класса электрического двигателя:

```
1   public class ElectricEngine
2   { }
3
4   public class Car
5   {
6   ElectricEngine engine;
7   public Car()
8   {
9   engine = new ElectricEngine();
10          }
11      }
```

При этом класс автомобиля полностью управляет жизненным циклом объекта двигателя. При уничтожении объекта автомобиля в области памяти вместе с ним будет уничтожен и объект двигателя. И в этом плане объект автомобиля является главным, а объект двигателя - зависимой.

На диаграммах UML отношение композиции проявляется в обычной стрелке от главной сущности к зависимой, при этом со стороны главной сущности, которая содержит, объект второй сущности, располагается закрашенный ромбик:



Агрегация

От композиции следует отличать агрегацию. Она также предполагает отношение **HAS A**, но реализуется она иначе:

```
1   public abstract class Engine
2   { }
3
4   public class Car
5   {
6   Engine engine;
7   public Car(Engine eng)
8   {
9   engine = eng;
10          }
11      }
```

При агрегации реализуется слабая связь, то есть в данном случае объекты Car и Engine будут равноправны. В конструктор Car передается ссылка на уже имеющийся объект Engine. И, как правило, определяется ссылка не на конкретный класс, а на абстрактный класс или интерфейс, что увеличивает гибкость программы.

Отношение агрегации на диаграммах UML отображается также, как и отношение композиции, только теперь ромбик будет незакрашенным:

При проектировании отношений между классами надо учитывать некоторые общие рекомендации. В частности, вместо наследования следует предпочитать композицию. При наследовании весь функционал класса-наследника жестко определен на этапе компиляции. И во время выполнения программы мы не можем его динамически переопределить. А класс-наследник не всегда может переопределить код, который определен в родительском классе. Композиция же позволяет динамически определять поведение объекта во время выполнения, и поэтому является более гибкой.

Вместо композиции следует предпочитать агрегацию, как более гибкий способ связи компонентов. В то же время не всегда агрегация уместна. Например, у нас есть класс человека, который содержит объект нервной системы. Понятно, что в реальности, по крайней мере на текущий момент, невозможно вовне определить нервную систему и внедрить ее в человека. То есть в данном случае человек будет главным компонентом, а нервная система - зависимым, подчиненным, и их создание и жизненный цикл будет происходить совместно, поэтому здесь лучше выбрать композицию.

Интерфейсы или абстрактные классы

Один из принципов проектирования гласит, что при создании системы классов надо программировать на уровне интерфейсов, а не их конкретных реализаций. Под интерфейсами в данном случае понимаются не только типы C#, определенные с помощью ключевого слова interface, а определение функционала без его конкретной реализации. То есть под данное определение попадают как собственно интерфейсы, так и абстрактные классы, которые могут иметь абстрактные методы без конкретной реализации.

В этом плане у абстрактных классов и интерфейсов много общего. Нередко при проектировании программ в паттернах мы можем заменять абстрактные классы на интерфейсы и наоборот. Однако все же они имеют некоторые отличия.

Когда следует использовать абстрактные классы:

- Если надо определить общий функционал для родственных объектов
- Если мы проектируем довольно большую функциональную единицу, которая содержит много базового функционал
- Если нужно, чтобы все производные классы на всех уровнях наследования имели некоторую общую реализацию. При использовании абстрактных классов, если мы захотим изменить базовый функционал во всех наследниках, то достаточно поменять его в абстрактном базовом классе.

Если же нам вдруг надо будет поменять название или параметры метода интерфейса, то придется вносить изменения и также во всех классы, которые данный интерфейс реализуют.

Когда следует использовать интерфейсы:

- Если нам надо определить функционал для группы разрозненных объектов, которые могут быть никак не связаны между собой.
- Если мы проектируем небольшой функциональный тип

Ключевыми здесь являются первые пункты, которые можно свести к следующему принципу: если классы относятся к единой системе классификации, то выбирается абстрактный класс. Иначе выбирается интерфейс. Посмотрим на примере.

Допустим, у нас есть система транспортных средств: легковой автомобиль, автобус, трамвай, поезд и т.д. Поскольку данные объекты являются родственными, мы можем выделить у них общие признаки, то в данном случае можно использовать абстрактные классы:

```
1  public abstract class Vehicle
2  {
3  public abstract void Move();
4      }
5
6  public class Car : Vehicle
7  {
8  public override void Move()
9  {
10 Console.WriteLine("Машина едет");
11         }
12     }
13
14     public class Bus : Vehicle
15     {
16         public override void Move()
17         {
18             Console.WriteLine("Автобус едет");
19         }
20     }
21
```

```
22  public class Tram : Vehicle
23  {
24  public override void Move()
25  {
26  Console.WriteLine("Трамвай едет");
27          }
28      }
```

Абстрактный класс Vehicle определяет абстрактный метод перемещения Move(), а классы-наследники его реализуют.

Но, предположим, что наша система транспорта не ограничивается вышеперечисленными транспортными средствами. Например, мы можем добавить самолеты, лодки. Возможно, также мы добавим лошадь - животное, которое может также выполнять роль транспортного средства. Также можно добавить дирижабль. Вобщем получается довольно широкий круг объектов, которые связаны только тем, что являются транспортным средством и должны реализовать некоторый метод Move(), выполняющий перемещение.

Так как объекты малосвязанные между собой, то для определения общего для всех них функционала лучше определить интерфейс. Тем более некоторые из этих объектов могут существовать в рамках параллельных систем классификаций. Например, лошадь может быть классом в структуре системы классов животного мира.

Возможная реализация интерфейса могла бы выглядеть следующим образом:

```
1   public interface IMovable
2   {
3   void Move();
4       }
5
6   public abstract class Vehicle
7   {}
8
9   public class Car : Vehicle, IMovable
10  {
11  public void Move()
12  {
13  Console.WriteLine("Машина едет");
14          }
15      }
16
17  public class Bus : Vehicle, IMovable
18  {
19  public void Move()
20  {
21  Console.WriteLine("Автобус едет");
22          }
23      }
24
25  public class Hourse : IMovable
26  {
27  public void Move()
28  {
29  Console.WriteLine("Лошадь скачет");
30          }
31      }
32
```

```
33  public class Aircraft : IMovable
34  {
35  public void Move()
36  {
37  Console.WriteLine("Самолет летит");
38          }
39      }
```

Теперь метод Move() определяется в интерфейсе IMovable, а конкретные классы его реализуют.

Говоря об использовании абстрактных классов и интерфейсов можно привести еще такую аналогию, как состояние и действие. Как правило, абстрактные классы фокусируются на общем состоянии классов-наследников. В то время как интерфейсы строятся вокруг какого-либо общего действия.

Например, солнце, костер, батарея отопления и электрический нагреватель выполняют функцию нагревания или излучения тепла. По большому счету выделение тепла - это единственный общий между ними признак. Можно ли для них создать общий абстрактный класс? Можно, но это не будет оптимальным решением, тем более у нас могут быть какие-то родственные сущности, которые мы, возможно, тоже захотим использовать. Поэтому для каждой вышеперечисленной сущности мы можем определить свою систему классификации. Например, в одной системе классов, которые наследуются от общего астрактного класса, были бы звезды, в том числе и солнце, планеты, астероиды и так далее - то есть все те объекты, которые могут иметь какое-то общее с солнцем состояние. В рамках другой системы классов мы могли бы определить электрические приборы, в том числе электронагреатель. И так, для каждой разноплановой сущности можно было бы составить свою систему классов, исходяющую от определенного абстрактного класса. А для общего действия определить интерфейс, например, IHeatable, в котором бы был метод Heat, и этот интерфейс реализовать во всех необходимых классах.

Таким образом, если разноплановые классы обладают каким-то общим действием, то это действие лучше выносить в интерфейс. А для одноплановых классов, которые имеют общее состояние, лучше определять абстрактный класс.

Практическая работа №16 Использование порождающих шаблонов

Фабричный метод (Factory Method)

Фабричный метод (Factory Method) - это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

Когда надо применять паттерн

- Когда заранее неизвестно, объекты каких типов необходимо создавать
- Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать.
- Когда создание новых объектов необходимо делегировать из базового класса классам наследникам

На языке UML паттерн можно описать следующим образом:

Формальное определение паттерна на языке C# может выглядеть следующим образом:

```
1   abstract class Product
2   {}
3
4   class ConcreteProductA : Product
5   {}
6
7   class ConcreteProductB : Product
8   {}
9
10  abstract class Creator
11  {
12  public abstract Product FactoryMethod();
13      }
14
15  class ConcreteCreatorA : Creator
16  {
17  public override Product FactoryMethod() { return new ConcreteProductA(); }
18      }
19
20  class ConcreteCreatorB : Creator
21  {
22  public override Product FactoryMethod() { return new ConcreteProductB(); }
23      }
```

Участники

- Абстрактный класс **Product** определяет интерфейс класса, объекты которого надо создавать.

Конкретные классы **ConcreteProductA** и **ConcreteProductB** представляют реализацию класса Product. Таких классов может быть множество

Абстрактный класс **Creator** определяет абстрактный фабричный метод FactoryMethod(), который возвращает объект Product.

Конкретные классы **ConcreteCreatorA** и **ConcreteCreatorB** - наследники класса Creator, определяющие свою реализацию метода FactoryMethod(). Причем метод FactoryMethod() каждого отдельного класса-создателя возвращает определенный конкретный тип продукта. Для каждого конкретного класса продукта определяется свой конкретный класс создателя.

Таким образом, класс Creator делегирует создание объекта Product своим наследникам. А классы ConcreteCreatorA и ConcreteCreatorB могут самостоятельно выбирать какой конкретный тип продукта им создавать.

Теперь рассмотрим на реальном примере. Допустим, мы создаем программу для сферы строительства. Возможно, вначале мы захотим построить многоэтажный панельный дом. И для этого выбирается соответствующий подрядчик, который возводит каменные дома. Затем нам захочется построить деревянный дом и для этого также надо будет выбрать нужного подрядчика:

```
class Program
{
static void Main(string[] args)
{
Developer dev = new PanelDeveloper("ООО КирпичСтрой");
House house2 = dev.Create(); 7
dev = new WoodDeveloper("Частный застройщик");
House house = dev.Create(); 10
            Console.ReadLine();
        }
    }
// абстрактный класс строительной компании
abstract class Developer
{
public string Name { get; set; } 18
public Developer (string n)
{
Name = n;
        }
// фабричный метод
abstract public House Create();
    }
// строит панельные дома
class PanelDeveloper : Developer
{
public PanelDeveloper(string n) : base(n)
{ }

public override House Create()
{
return new PanelHouse();
        }
    }
// строит деревянные дома
class WoodDeveloper : Developer
{
public WoodDeveloper(string n) : base(n)
{ }

public override House Create()
{
```

```
45              return new WoodHouse();
46          }
47      }
48
49  abstract class House
50  { }
51
52  class PanelHouse : House
53  {
54  public PanelHouse()
55  {
56  Console.WriteLine("Панельный дом построен");
57          }
58      }
59  class WoodHouse : House
60  {
61  public WoodHouse()
62  {
63  Console.WriteLine("Деревянный дом построен");
64          }
65      }
```

В качестве абстрактного класса Product здесь выступает класс House. Его две конкретные реализации - PanelHouse и WoodHouse представляют типы домов, которые будут строить подрядчики. В качестве абстрактного класса создателя выступает Developer, определяющий абстрактный метод Create(). Этот метод реализуется в классах-наследниках WoodDeveloper и PanelDeveloper. И если в будущем нам потребуется построить дома какого-то другого типа, например, кирпичные, то мы можем с легкостью создать новый класс кирпичных домов, унаследованный от House, и определить класс соответствующего подрядчика. Таким образом, система получится легко расширяемой. Правда, недостатки паттерна тоже очевидны - для каждого нового продукта необходимо создавать свой класс создателя.

Абстрактная фабрика (Abstract Factory)

Паттерн "Абстрактная фабрика" (Abstract Factory) предоставляет интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов.

Когда использовать абстрактную фабрику

Когда система не должна зависеть от способа создания и компоновки новых объектов
Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными
С помощью UML абстрактную фабрику можно представить следующим образом:

Формальное определение паттерна на языке C# может выглядеть следующим образом:

```
1       abstract class AbstractFactory
2       {
3           public abstract AbstractProductA CreateProductA();
4           public abstract AbstractProductB CreateProductB();
5       }
6       class ConcreteFactory1: AbstractFactory
7       {
8           public override AbstractProductA CreateProductA()
9           {
```

```
10          return new ProductA1();
11      }
12
13      public override AbstractProductB CreateProductB()
14      {
15          return new ProductB1();
16      }
17  }
18  class ConcreteFactory2: AbstractFactory
19  {
20      public override AbstractProductA CreateProductA()
21      {
22          return new ProductA2();
23      }
24
25      public override AbstractProductB CreateProductB()
26      {
27          return new ProductB2();
28      }
29  }
30
31  abstract class AbstractProductA
32  {}
33
34  abstract class AbstractProductB
35  {}
36
37  class ProductA1: AbstractProductA
38  {}
39
40  class ProductB1: AbstractProductB
41  {}
42
43  class ProductA2: AbstractProductA
44  {}
45
46  class ProductB2: AbstractProductB
47  {}
48
49  class Client
50  {
51      private AbstractProductA abstractProductA; pri-
52      vate AbstractProductB abstractProductB;
53
54      public Client(AbstractFactory factory)
55      {
56          abstractProductB = factory.CreateProductB();
57          abstractProductA = factory.CreateProductA();
58      }
59      public void Run()
60      { }
61  }
```

Паттерн определяет следующих участников:

- Абстрактные классы **AbstractProductA** и **AbstractProductB** определяют интерфейс для классов, объекты которых будут создаваться в программе.
- Конкретные классы **ProductA1** / **ProductA2** и **ProductB1** / **ProductB2** представляют конкретную реализацию абстрактных классов
- Абстрактный класс фабрики **AbstractFactory** определяет методы для создания объектов. Причем методы возвращают абстрактные продукты, а не их конкретные реализации.
- Конкретные классы фабрик **ConcreteFactory1** и **ConcreteFactory2** реализуют абстрактные методы базового класса и непосредственно определяют какие конкретные продукты использовать
- Класс клиента **Client** использует класс фабрики для создания объектов. При этом он использует исключительно абстрактный класс фабрики AbstractFactory и абстрактные классы продуктов AbstractProductA и AbstractProductB и никак не зависит от их конкретных реализаций

Посмотрим, как мы можем применить паттерн. Например, мы делаем игру, где пользователь должен управлять некими супергероями, при этом каждый супергерой имеет определенное оружие и определенную модель передвижения. Различные супергерои могут определяться комплексом признаков. Например, эльф может летать и должен стрелять из арбалета, другой супергерой должен бегать и управлять мечом. Таким образом, получается, что сущность оружия и модель передвижения являются взаимосвязанными и используются в комплексе. То есть имеется один из доводов в пользу использования абстрактной фабрики.

И кроме того, наша задача при проектировании игры абстрагировать создание супергероев от самого класса супергероя, чтобы создать более гибкую архитектуру. И для этого применим абстрактную фабрику:

```
1   class Program
2   {
3   static void Main(string[] args)
4   {
5   Hero elf = new Hero(new ElfFactory());
6   elf.Hit();
7   elf.Run();
8
9   Hero voin = new Hero(new VoinFactory());
10  voin.Hit();
11  voin.Run();
12
13              Console.ReadLine();
14          }
15      }
16  //абстрактный класс - оружие
17  abstract class Weapon
18  {
19  public abstract void Hit();
20      }
21  // абстрактный класс движение
22  abstract class Movement
23  {
24  public abstract void Move();
25      }
26
```

```
27      // класс арбалет
28      class Arbalet : Weapon
29      {
30          public override void Hit()
31          {
32              Console.WriteLine("Стреляем из арбалета");
33          }
34      }
35      // класс меч
36      class Sword : Weapon
37      {
38          public override void Hit()
39          {
40              Console.WriteLine("Бьем мечом");
41          }
42      }
43      // движение полета
44      class FlyMovement : Movement
45      {
46          public override void Move()
47          {
48              Console.WriteLine("Летим");
49          }
50      }
51      // движение - бег
52      class RunMovement : Movement
53      {
54          public override void Move()
55          {
56              Console.WriteLine("Бежим");
57          }
58      }
59      // класс абстрактной фабрики ab-
60      stract class HeroFactory
61      {
62          public abstract Movement CreateMovement();
63          public abstract Weapon CreateWeapon();
64      }
65      // Фабрика создания летящего героя с арбалетом
66      class ElfFactory : HeroFactory
67      {
68          public override Movement CreateMovement()
69          {
70              return new FlyMovement();
71          }
72
73          public override Weapon CreateWeapon()
74          {
75                  return new Arbalet();
76          }
77      }
78      // Фабрика создания бегущего героя с мечом
```

```
79  class VoinFactory : HeroFactory
80  {
81  public override Movement CreateMovement()
82  {
83  return new RunMovement();
84          }
85
86  public override Weapon CreateWeapon()
87  {
88  return new Sword();
89          }
90      }
91  // клиент - сам супергерой
92  class Hero
93  {
94  private Weapon weapon;
95  private Movement movement;
96  public Hero(HeroFactory factory)
97  {
98  weapon = factory.CreateWeapon();
99  movement = factory.CreateMovement();
100         }
101 public void Run()
102 {
103 movement.Move();
104         }
105 public void Hit()
106 {
107 weapon.Hit();
108         }
109     }
```

Таким образом, создание супергероя абстрагируется от самого класса супергероя. В то же время нельзя не отметить и недостатки шаблона. В частности, если нам захочется добавить в конфигурацию супергероя новый объект, например, тип одежды, то придется переделывать классы фабрик и класс супергероя. Поэтому возможности по расширению в данном паттерне имеют некоторые ограничения.

Одиночка

Одиночка (Singleton, Синглтон) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Когда надо использовать Синглтон? Когда необходимо, чтобы для класса существовал только один экземпляр

Синглтон позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие синглтона от глобальных переменных.

Классическая реализация данного шаблона проектирования на C# выглядит следующим образом:

```
1   class Singleton
2   {
3   private static Singleton instance; 4
5   private Singleton()
6   {}
```

```
7
8           public static Singleton getInstance()
9           {
10              if (instance == null)
11                  instance = new Singleton();
12              return instance;
13          }
14      }
```

В классе определяется статическая переменная - ссылка на конкретный экземпляр данного объекта и приватный конструктор. В статическом методе getInstance() этот конструктор вызывается для создания объекта, если, конечно, объект отсутствует и равен null.

Для применения паттерна Одиночка создадим небольшую программу. Например, на каждом компьютере можно одномоментно запустить только одну операционную систему. В этом плане операционная система будет реализоваться через паттерн синглтон:

```
1   class Program
2   {
3   static void Main(string[] args)
4   {
5   Computer comp = new Computer();
6   comp.Launch("Windows 8.1");
7   Console.WriteLine(comp.OS.Name);
8
9   // у нас не получится изменить ОС, так как объект уже создан
10  comp.OS = OS.getInstance("Windows 10");
11  Console.WriteLine(comp.OS.Name);
12
13              Console.ReadLine();
14          }
15      }
16  class Computer
17  {
18  public OS OS { get; set; }
19  public void Launch(string osName)
20  {
21  OS = OS.getInstance(osName);
22          }
23      }
24  class OS
25  {
26  private static OS instance; 27
28          public string Name { get; private set; }
29
30  protected OS(string name)
31  {
32  this.Name=name;
33          }
34
35  public static OS getInstance(string name)
36  {
37  if (instance == null)
38  instance = new OS(name);
```

```
            return instance;
        }
    }
```

Синглтон и многопоточность

При применении паттерна синглтон в многопоточным программах мы можем столкнуться с проблемой, которую можно описать следующим образом:

```
static void Main(string[] args)
{
(new Thread(() =>
{
Computer comp2 = new Computer();
comp2.OS = OS.getInstance("Windows 10");
Console.WriteLine(comp2.OS.Name);

        })).Start();

Computer comp = new Computer();
comp.Launch("Windows 8.1");
Console.WriteLine(comp.OS.Name);
Console.ReadLine();
    }
```

Здесь запускается дополнительный поток, который получает доступ к синглтону. Параллельно выполняется тот код, который идет запуска потока и кторый также обращается к синглтону. Таким образом, и главный, и дополнительный поток пытаются инициализировать синглтон нужным значением - "Windows 10", либо "Windows 8.1". Какое значение сиглтон получит в итоге, пресказать в данном случае невозможно.

Вывод программы может быть такой:

```
Windows 8.1
Windows 10
```

Или такой:

```
Windows 8.1
Windows 8.1
```

В итоге мы сталкиваемся с проблемой инициализации синглтона, когда оба потока одновременно обращаются к коду:

```
if (instance == null)
    instance = new OS(name);
```

Чтобы решить эту проблему, перепишем класс синглтона следующим образом:

```
class OS
{
    private static OS instance;

    public string Name { get; private set; }  pri-
    vate static object syncRoot = new Object();

    protected OS(string name)
    {
        this.Name = name;
    }

    public static OS getInstance(string name)
    {
        if (instance == null)
```

```
            {
                lock (syncRoot)
                {
                    if (instance == null)
                        instance = new OS(name);
                }
            }
            return instance;
        }
    }
```

Чтобы избежать одновременного доступа к коду из разных потоков критическая секция заключается в блок **lock**.

Другие реализации синглтона

Выше были рассмотрены общие стандартные реализации: потоконебезопасная и потокобезопасная реализации паттерна. Но есть еще ряд дополнительных реализаций, которые можно рассмотреть.

Потокобезопасная реализация без использования lock

```
public class Singleton
{
private static readonly Singleton instance = new Singleton(); 4
        public string Date { get; private set; }

private Singleton()
{
Date = System.DateTime.Now.TimeOfDay.ToString();
        }

public static Singleton GetInstance()
{
return instance;
        }
    }
```

Данная реализация также потокобезопасная, то есть мы можем использовать ее в потоках так:

```
(new Thread(() =>
{
Singleton singleton1 = Singleton.GetInstance();
Console.WriteLine(singleton1.Date);
})).Start();

Singleton singleton2 = Singleton.GetInstance();
Console.WriteLine(singleton2.Date);
```

Lazy-реализация

Определение объекта синглтона в виде статического поля класса открывает нам дорогу к созданию Lazy-реализации паттерна Синглтон, то есть такой реализации, где данные будут инициализироваться только перед непосредственным использованием. Поскольку статические поля инициализируются перед первым доступом к статическому членам класса и перед вызовом статического конструктора (при его наличии). Однако здесь мы можем столкнуться с двумя трудностями.

Во-первых, класс синглтона может иметь множество статических переменных. Возможно, мы вообще не будем обращаться к объекту синглтона, а будем использовать какие-то другие статические переменные:

```
1  public class Singleton
2  {
3  private static readonly Singleton instance = new Singleton();
4  public static string text = "hello";
5  public string Date { get; private set; } 6
7  private Singleton()
8  {
9  Console.WriteLine($"Singleton ctor {DateTime.Now.TimeOfDay}");
10 Date = System.DateTime.Now.TimeOfDay.ToString();
11         }
12
13 public static Singleton GetInstance()
14 {
15 Console.WriteLine($"GetInstance {DateTime.Now.TimeOfDay}");
16 Thread.Sleep(500);
17 return instance;
18         }
19       }
20 class Program
21 {
22 static void Main(string[] args)
23 {
24 Console.WriteLine($"Main {DateTime.Now.TimeOfDay}");
25 Console.WriteLine(Singleton.text);
26 Console.Read();
27         }
28       }
```

В данном случае идет только обращение к переменной text, однако статическое поле instance также будет инициализировано. Например, консольный вывод в данном случае мог бы выглядеть следующим образом:

```
Singleton ctor 16:05:54.1469982
Main 16:05:54.2920316
hello
```

В данном случае мы видим, что статическое поле instance инициализировано.

Для решения этой проблемы выделим отдельный внутренний класс в рамках класса синглтона:

```
1  public class Singleton
2  {
3  public string Date { get; private set; }
4  public static string text = "hello";
5  private Singleton()
6  {
7  Console.WriteLine($"Singleton ctor {DateTime.Now.TimeOfDay}");
8  Date = DateTime.Now.TimeOfDay.ToString();
9          }
10
11 public static Singleton GetInstance()
12 {
```

```
13            Console.WriteLine($"GetInstance {DateTime.Now.TimeOfDay}");
14            return Nested.instance;
15        }
16
17        private class Nested
18        {
19            internal static readonly Singleton instance = new Singleton();
20        }
21    }
22    class Program
23    {
24        static void Main(string[] args)
25        {
26            Console.WriteLine($"Main {DateTime.Now.TimeOfDay}");
27            Console.WriteLine(Singleton.text);
28            Console.Read();
29        }
30    }
```

Теперь статическая переменная, которая представляет объект синглтона, определена во вложенном классе Nested. Чтобы к этой переменной можно было обращаться из класса синглтона, она имеет модификатор internal, в то же время сам класс Nested имеет модификатор private, что позволяет гарантировать, что данный класс будет доступен только из класса Singleton.

Консольный вывод в данном случае мог бы выглядеть следующим образом:

```
Main 16:11:40.1320873
hello
```

Далее мы сталкиваемся со второй проблемой: статические поля инициализируются перед первым доступом к статическому членам класса и перед вызовом статического конструктора (при его наличии). Но когда именно? Если класс содержит статические поля, не содержит статического конструктора, то время инициализации статических полей зависит от реализации платформы. Нередко это непосредственно перед первым использованием, но тем не менее момент точно не определен - это может быть происходить и чуть раньше. Однако если класс содержит статический конструктор, то статические поля будут инициализироваться непосредственно либо при создании первого экземпляра класса, либо при первом обращении к статическим членам класса.

Например, рассмотрим выполнение следующей программы:

```
1    static void Main(string[] args)
2    {
3    Console.WriteLine($"Main {DateTime.Now.TimeOfDay}");
4    Console.WriteLine(Singleton.text);
5
6    Singleton singleton1 = Singleton.GetInstance();
7    Console.WriteLine(singleton1.Date);
8    Console.Read();
9     }
```

Ее возможный консольный вывод:

```
Main 16:33:33.1404818
hello
Singleton ctor 16:33:33.1564802
GetInstance 16:33:33.1574824
16:33:33.1564802
```

Мы видим, что код метода GetInstance, который идет до вызова конструктора класса Singleton, выполняется после выполнения этого конструктора. Поэтому добавим в выше определенный класс Nested статический конструктор:

```
1   public class Singleton
2   {
3   public string Date { get; private set; }
4   public static string text = "hello";
5   private Singleton()
6   {
7   Console.WriteLine($"Singleton ctor {DateTime.Now.TimeOfDay}");
8   Date = DateTime.Now.TimeOfDay.ToString();
9           }
10
11  public static Singleton GetInstance()
12  {
13  Console.WriteLine($"GetInstance {DateTime.Now.TimeOfDay}");
14  Thread.Sleep(500);
15  return Nested.instance;
16          }
17
18  private class Nested
19  {
20  static Nested() { }
21  internal static readonly Singleton instance = new Singleton();
22          }
23      }
```

Теперь при выполнении той же программы мы получим полноценную Lazy-реализацию:

```
Main 16:37:18.4108064
hello
GetInstance 16:37:18.4208062
Singleton ctor 16:37:18.4218065
16:37:18.4228061
```

Реализация через класс Lazy<T>

Еще один способ создания синглтона представляет использование класса Lazy<T>:

```
1   public class Singleton
2   {
3   private static readonly Lazy<Singleton> lazy =
4   new Lazy<Singleton>(() => new Singleton()); 5
6           public string Name { get; private set; }
7
8   private Singleton()
9   {
10  Name = System.Guid.NewGuid().ToString();
11          }
12
13  public static Singleton GetInstance()
14  {
15  return lazy.Value;
16          }
17      }
```

Практическая работа №17 Использование структурных шаблонов

Декоратор (Decorator)

Декоратор (Decorator) представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Для определения нового функционала в классах нередко используется наследование. Декораторы же предоставляет наследованию более гибкую альтернативу, поскольку позволяют динамически в процессе выполнения определять новые возможности у объектов.

Когда следует использовать декораторы?

Когда надо динамически добавлять к объекту новые функциональные возможности. При этом данные возможности могут быть сняты с объекта

Когда применение наследования неприемлемо. Например, если нам надо определить множество различных функциональностей и для каждой функциональности наследовать отдельный класс, то структура классов может очень сильно разрастись. Еще больше она может разрастись, если нам необходимо создать классы, реализующие все возможные сочетания добавляемых функциональностей.

Схематически шаблон "Декоратор" можно выразить следующим образом:

Формальная организация паттерна в C# могла бы выглядеть следующим образом:

```
1   abstract class Component
2   {
3       public abstract void Operation();
4   }
5   class ConcreteComponent : Component
6   {
7       public override void Operation()
8       {}
9   }
10  abstract class Decorator : Component
11  {
12      protected Component component;
13
14      public void SetComponent(Component component)
15      {
16          this.component = component;
17      }
18
19      public override void Operation()
20      {
21          if (component != null) component.Opera-
22              tion();
23      }
24  }
25  class ConcreteDecoratorA : Decorator
26  {
27      public override void Operation()
28      {
29          base.Operation();
30      }
31  }
32  class ConcreteDecoratorB : Decorator
33  {
```

```
        public override void Operation()
        {
            base.Operation();
        }
    }
```

Участники

- **Component**: абстрактный класс, который определяет интерфейс для наследуемых объектов

ConcreteComponent: конкретная реализация компонента, в которую с помощью декоратора добавляется новая функциональность

Decorator: собственно декоратор, реализуется в виде абстрактного класса и имеет тот же базовый класс, что и декорируемые объекты. Поэтому базовый класс Component должен быть по возможности легким и определять только базовый интерфейс.

Класс декоратора также хранит ссылку на декорируемый объект в виде объекта базового класса Component и реализует связь с базовым классом как через наследование, так и через отношение агрегации.

Классы **ConcreteDecoratorA** и **ConcreteDecoratorB** представляют дополнительные функциональности, которыми должен быть расширен объект ConcreteComponent.

Рассмотрим пример. Допустим, у нас есть пиццерия, которая готовит различные типы пицц с различными добавками. Есть итальянская, болгарская пиццы. К ним могут добавляться помидоры, сыр и т.д. И в зависимости от типа пицц и комбинаций добавок пицца может иметь разную стоимость. Теперь посмотрим, как это изобразить в программе на C#:

```
class Program
{
static void Main(string[] args)
{
Pizza pizza1 = new ItalianPizza();
pizza1 = new TomatoPizza(pizza1); // итальянская пицца с томатами
Console.WriteLine("Название: {0}", pizza1.Name);
Console.WriteLine("Цена: {0}", pizza1.GetCost());

Pizza pizza2 = new ItalianPizza();
pizza2 = new CheesePizza(pizza2);// итальянская пиццы с сыром
Console.WriteLine("Название: {0}", pizza2.Name);
Console.WriteLine("Цена: {0}", pizza2.GetCost());

Pizza pizza3 = new BulgerianPizza();
pizza3 = new TomatoPizza(pizza3);
pizza3 = new CheesePizza(pizza3);// болгарская пиццы с томатами и сыром
Console.WriteLine("Название: {0}", pizza3.Name);
Console.WriteLine("Цена: {0}", pizza3.GetCost());

            Console.ReadLine();
        }
    }

abstract class Pizza
{
public Pizza(string n)
{
```

```
29          this.Name = n;
30      }
31      public string Name {get; protected set;} pub-
32      lic abstract int GetCost();
33  }
34
35  class ItalianPizza : Pizza
36  {
37      public ItalianPizza() : base("Итальянская пицца")
38      { }
39      public override int GetCost()
40      {
41          return 10;
42      }
43  }
44  class BulgerianPizza : Pizza
45  {
46      public BulgerianPizza()
47          : base("Болгарская пицца")
48      { }
49      public override int GetCost()
50      {
51          return 8;
52      }
53  }
54
55  abstract class PizzaDecorator : Pizza
56  {
57      protected Pizza pizza;
58      public PizzaDecorator(string n, Pizza pizza) : base(n)
59      {
60          this.pizza = pizza;
61      }
62  }
63
64  class TomatoPizza : PizzaDecorator
65  {
66      public TomatoPizza(Pizza p)
67          : base(p.Name + ", с томатами", p)
68      { }
69
70      public override int GetCost()
71      {
72          return pizza.GetCost() + 3;
73      }
74  }
75
76  class CheesePizza : PizzaDecorator
77  {
78      public CheesePizza(Pizza p)
79          : base(p.Name + ", с сыром", p)
80      { }
```

```
81
82          public override int GetCost()
83          {
84              return pizza.GetCost() + 5;
85          }
86      }
```

В качестве компонента здесь выступает абстрактный класс Pizza, который определяет базовую функциональность в виде свойства Name и метода GetCost(). Эта функциональность реализуется двумя подклассами ItalianPizza и BulgerianPizza, в которых жестко закодированы название пиццы и ее цена.

Декоратором является абстрактный класс PizzaDecorator, который унаследован от класса Pizza и содержит ссылку на декорируемый объект Pizza. В отличие от формальной схемы здесь установка декорируемого объекта происходит не в методе SetComponent, а в конструкторе.

Отдельные функциональности - добавление томатов и сыры к пиццам реализованы через классы TomatoPizza и CheesePizza, которые обертывают объект Pizza и добавляют к его имени название добавки, а к цене - стоимость добавки, то есть переопределяя метод GetCost и изменяя значение свойства Name.

Благодаря этому при создании пиццы с добавками произойдет ее обертывание декоратором:

```
1   Pizza pizza3 = new BulgerianPizza();
2   pizza3 = new TomatoPizza(pizza3);
3   pizza3 = new CheesePizza(pizza3);
```

Сначала объект BulgerianPizza обертывается декоратором TomatoPizza, а затем CheesePizza. И таких обертываний мы можем сделать множество. Просто достаточно унаследовать от декоратора класс, который будет определять дополнительный функционал.

А если бы мы использовали наследование, то в данном случае только для двух видов пицц с двумя добавками нам бы пришлось создать восемь различных классов, которые бы описывали все возможные комбинации. Поэтому декораторы являются более предпочтительным в данном случае методом.

Адаптер (Adapter)

Паттерн Адаптер (Adapter) предназначен для преобразования интерфейса одного класса в интерфейс другого. Благодаря реализации данного паттерна мы можем использовать вместе классы с несовместимыми интерфейсами.

Когда надо использовать Адаптер?

Когда необходимо использовать имеющийся класс, но его интерфейс не соответствует потребностям

Когда надо использовать уже существующий класс совместно с другими классами, интерфейсы которых не совместимы

Формальное определение паттерна на UML выглядит следующим образом:

Формальное описание адаптера объектов на C# выглядит таким образом:

```
1       class Client
2       {
3           public void Request(Target target)
4           {
5               target.Request();
6           }
7       }
8       // класс, к которому надо адаптировать другой класс
```

```
9       class Target
10      {
11          public virtual void Request()
12          {}
13      }
14
15  // Адаптер
16  class Adapter : Target
17  {
18  private Adaptee adaptee = new Adaptee(); 19
20  public override void Request()
21  {
22  adaptee.SpecificRequest();
23              }
24          }
25
26  // Адаптируемый класс
27  class Adaptee
28  {
29  public void SpecificRequest()
30  {}
31          }
```

Участники

- **Target**: представляет объекты, которые используются клиентом
- **Client**: использует объекты Target для реализации своих задач
- **Adaptee**: представляет адаптируемый класс, который мы хотели бы использовать у клиента вместо объектов Target

Adapter: собственно адаптер, который позволяет работать с объектами Adaptee как с объектами Target.

То есть клиент ничего не знает об Adaptee, он знает и использует только объекты Target. И благодаря адаптеру мы можем на клиенте использовать объекты Adaptee как Target

Теперь разберем реальный пример. Допустим, у нас есть путешественник, который путешествует на машине. Но в какой-то момент ему приходится передвигаться по пескам пустыни, где он не может ехать на машине. Зато он может использовать для передвижения верблюда. Однако в классе путешественника использование класса верблюда не предусмотрено, поэтому нам надо использовать адаптер:

```
1   class Program
2   {
3   static void Main(string[] args)
4   {
5   // путешественник
6   Driver driver = new Driver();
7   // машина
8   Auto auto = new Auto();
9   // отправляемся в путешествие
10  driver.Travel(auto);
11  // встретились пески, надо использовать верблюда
12  Camel camel = new Camel();
13  // используем адаптер
14  ITransport camelTransport = new CamelToTransportAdapter(camel);
15  // продолжаем путь по пескам пустыни
16  driver.Travel(camelTransport);
```

```
17
18                  Console.Read();
19  }
20      }
21  interface ITransport
22  {
23  void Drive();
24      }
25  // класс машины
26  class Auto : ITransport
27  {
28  public void Drive()
29  {
30  Console.WriteLine("Машина едет по дороге");
31          }
32      }
33  class Driver
34  {
35  public void Travel(ITransport transport)
36  {
37  transport.Drive();
38          }
39      }
40  // интерфейс животного
41  interface IAnimal
42  {
43  void Move();
44      }
45  // класс верблюда
46  class Camel : IAnimal
47  {
48  public void Move()
49  {
50  Console.WriteLine("Верблюд идет по пескам пустыни");
51          }
52      }
53  // Адаптер от Camel к ITransport
54  class CamelToTransportAdapter : ITransport
55  {
56  Camel camel;
57  public CamelToTransportAdapter(Camel c)
58  {
59  camel = c;
60          }
61
62  public void Drive()
63  {
64  camel.Move();
65          }
66      }
```

И консоль выведет:

```
Машина едет по дороге
```

Верблюд идет по пескам пустыни

В данном случае в качестве клиента применяется класс Driver, который использует объект ITransport. Адаптируемым является класс верблюда Camel, который нужно использовать в качестве объекта ITransport. И адптером служит класс CamelToTransportAdapter.

Возможно, кому-то покажется надуманной проблема использования адаптеров особенно в данном случае, так как мы могли бы применить интерфейс ITransport к классу Camel и реализовать его метод Drive(). Однако, в данном случае может случиться дублирование функциональностей: интерфейс IAnimal имеет метод Move(), реализация которого в классе верблюда могла бы быть похожей на реализацию метода Drive() из интерфейса ITransport. Кроме того, нередко бывает, что классы спроектированы кем-то другим, и мы никак не можем на них повлиять. Мы только используем их. В результате чего адаптеры довольно широко распространены в .NET. В частности, многочисленные встроенные классы, которые используются для подключения к различным системам баз данных, как раз и реализуют паттерн адаптер (например, класс System.Data.SqlClient.SqlDataAdapter). **Фасад (Facade)**

Фасад (Facade) представляет шаблон проектирования, который позволяет скрыть сложность системы с помощью предоставления упрощенного интерфейса для взаимодействия с ней.

Когда использовать фасад?

- Когда имеется сложная система, и необходимо упростить с ней работу. Фасад позволит определить одну точку взаимодействия между клиентом и системой.
- Когда надо уменьшить количество зависимостей между клиентом и сложной системой. Фасадные объекты позволяют отделить, изолировать компоненты системы от клиента и развивать и работать с ними независимо.
- Когда нужно определить подсистемы компонентов в сложной системе. Создание фасадов для компонентов каждой отдельной подсистемы позволит упростить взаимодействие между ними и повысить их независимость друг от друга.

В UML общую схему фасада можно представить следующим образом:

Формальное определение программы в C# может выглядеть так:

```
1   class SubsystemA
2   {
3      public void A1()
4      {}
5   }
6   class SubsystemB
7   {
8      public void B1()
9      {}
10  }
11  class SubsystemC
12  {
13     public void C1()
14     {}
15  }
16
17  public class Facade
18  {
19     SubsystemA subsystemA;
20     SubsystemB subsystemB;
21     SubsystemC subsystemC;
```

```
22
23          public Facade(SubsystemA sa, SubsystemB sb, SubsystemC sc)
24          {
25             subsystemA = sa;
26             subsystemB = sb;
27             subsystemC = sc;
28          }
29          public void Operation1()
30          {
31             subsystemA.A1();
32             subsystemB.B1();
33             subsystemC.C1();
34          }
35          public void Operation2()
36          {
37             subsystemB.B1();
38             subsystemC.C1();
39          }
40       }
41
   42  class Client
   43  {
   44  public void Main()
   45  {
   46  Facade facade = new Facade(new SubsystemA(), new SubsystemB(), new SubsystemC());
   47  facade.Operation1();
   48  facade.Operation2();
49          }
50       }
```

Участники

- Классы SubsystemA, SubsystemB, SubsystemC и т.д. являются компонентами сложной подсистемы, с которыми должен взаимодействовать клиент

Client взаимодействует с компонентами подсистемы

Facade - непосредственно фасад, который предоставляет интерфейс клиенту для работы с компонентами

Рассмотрим применение паттерна в реальной задаче. Думаю, большинство программистов согласятся со мной, что писать в Visual Studio код одно удовольствие по сравнению с тем, как писался код ранее до появления интегрированных сред разработки. Мы просто пишем код, нажимаем на кнопку и все - приложение готово. В данном случае интегрированная среда разработки представляет собой фасад, который скрывает всю сложность процесса компиляции и запуска приложения. Теперь опишем этот фасад в программе на C#:

```
1   class Program
2   {
3   static void Main(string[] args)
4   {
5   TextEditor textEditor = new TextEditor();
6   Compiller compiller = new Compiller();
7   CLR clr = new CLR(); 8
9               VisualStudioFacade ide = new VisualStudioFacade(textEditor, compiller,
clr); 10
11              Programmer programmer = new Programmer();
```

```
12          programmer.CreateApplication(ide);
13
14          Console.Read();
15      }
16  }
17  // текстовый редактор
18  class TextEditor
19  {
20      public void CreateCode()
21      {
22          Console.WriteLine("Написание кода");
23      }
24      public void Save()
25      {
26          Console.WriteLine("Сохранение кода");
27      }
28  }
29  class Compiller
30  {
31      public void Compile()
32      {
33          Console.WriteLine("Компиляция приложения");
34      }
35  }
36  class CLR
37  {
38      public void Execute()
39      {
40          Console.WriteLine("Выполнение приложения");
41      }
42      public void Finish()
43      {
44          Console.WriteLine("Завершение работы приложения");
45      }
46  }
47
48  class VisualStudioFacade
49  {
50      TextEditor textEditor;
51      Compiller compiller;
52      CLR clr;
53      public VisualStudioFacade(TextEditor te, Compiller compil, CLR clr)
54      {
55          this.textEditor = te; this.com-
56          piller = compil; this.clr = clr;
57      }
58      public void Start()
59      {
60          textEditor.CreateCode();
61          textEditor.Save(); compiller.-
62          Compile();
63
```

```
64              clr.Execute();
65          }
66          public void Stop()
67          {
68              clr.Finish();
69          }
70      }
71
72  class Programmer
73  {
74  public void CreateApplication(VisualStudioFacade facade)
75  {
76  facade.Start();
77  facade.Stop();
78          }
79      }
```

В данном случае компонентами системы являются класс текстового редактора TextEditor, класс компилятора Compiller и класс общеязыковой среды выполнения CLR. Клиентом выступает класс программиста, фасадом - класс VisualStudioFacade, который через свои методы делегирует выполнение работы компонентам и их методам.

При этом надо учитывать, что клиент может при необходимости обращаться напрямую к компонентам, например, отдельно от других компонентов использовать текстовый редактор. Но в виду сложности процесса создания приложения лучше использовать фасад. Также это не единственный возможный фасад для работы с данными компонентами. При необходимости можно создавать альтернативные фасады также, как в реальной жизни мы можем использовать альтернативные среды разработки.

Практическая работа №18 Использование поведенческих шаблонов

Паттерны поведения

Стратегия (Strategy)

обеспечивает их взаимозаменяемость. В зависимости от ситуации мы можем легко заменить один используемый алгоритм другим. При этом замена алгоритма происходит независимо от объекта, который использует данный алгоритм.

Когда использовать стратегию?

- Когда есть несколько родственных классов, которые отличаются поведением. Можно задать один основной класс, а разные варианты поведения вынести в отдельные классы и при необходимости их применять
- Когда необходимо обеспечить выбор из нескольких вариантов алгоритмов, которые можно легко менять в зависимости от условий
- Когда необходимо менять поведение объектов на стадии выполнения программы
- Когда класс, применяющий определенную функциональность, ничего не должен знать о ее реализации

Формально паттерн Стратегия можно выразить следующей схемой UML:

Формальное определение паттерна на языке C# может выглядеть следующим образом:

```
1   public interface IStrategy
2   {
3       void Algorithm();
4   }
5
6   public class ConcreteStrategy1 : IStrategy
7   {
8       public void Algorithm()
9       {}
10  }
11
12  public class ConcreteStrategy2 : IStrategy
13  {
14      public void Algorithm()
15      {}
16  }
17
18  public class Context
19  {
20      public IStrategy ContextStrategy { get; set; }
21
22      public Context(IStrategy _strategy)
23      {
24          ContextStrategy = _strategy;
25      }
26
27      public void ExecuteAlgorithm()
28      {
29          ContextStrategy.Algorithm();
30      }
31  }
```

Участники

Как видно из диаграммы, здесь есть следующие участники:

- Интерфейс IStrategy, который определяет метод Algorithm(). Это общий интерфейс для всех реализующих его алгоритмов. Вместо интерфейса здесь также можно было бы использовать абстрактный класс.

Классы ConcreteStrategy1 и ConcreteStrategy, которые реализуют интерфейс IStrategy, предоставляя свою версию метода Algorithm(). Подобных классов-реализаций может быть множество.

Класс Context хранит ссылку на объект IStrategy и связан с интерфейсом IStrategy отношением агрегации.

В данном случае объект IStrategy заключена в свойстве ContextStrategy, хотя также для нее можно было бы определить приватную переменную, а для динамической установки использовать специальный метод.

Теперь рассмотрим конкретный пример. Существуют различные легковые машины, которые используют разные источники энергии: электричество, бензин, газ и так далее. Есть гибридные автомобили. В целом они похожи и отличаются преимущественно видом источника энергии. Не говоря уже о том, что мы можем изменить применяемый источник энергии, модифицировав автомобиль. И в данном случае вполне можно применить паттерн стратегию:

```
class Program
{
static void Main(string[] args)
{
Car auto = new Car(4, "Volvo", new PetrolMove());
auto.Move();
auto.Movable = new ElectricMove();
auto.Move();

            Console.ReadLine();
        }
    }
interface IMovable
{
void Move();
    }

class PetrolMove : IMovable
{
public void Move()
{
Console.WriteLine("Перемещение на бензине");
        }
    }

class ElectricMove : IMovable
{
public void Move()
{
Console.WriteLine("Перемещение на электричестве");
        }
    }
class Car
{
protected int passengers; // кол-во пассажиров
```

```
36          protected string model; // модель автомобиля
37
38          public Car(int num, string model, IMovable mov)
39          {
40              this.passengers = num;
41              this.model = model;
42              Movable = mov;
43          }
44          public IMovable Movable { private get; set; }
45          public void Move()
46          {
47              Movable.Move();
48          }
49      }
```

В данном случае в качестве IStrategy выступает интерфейс IMovable, определяющий метод Move(). А реализующий этот интерфейс семейство алгоритмов представлено классами ElectricMove и PetroleMove. И данные алгоритмы использует класс Car.

Наблюдатель (Observer)

Паттерн "Наблюдатель" (Observer) представляет поведенческий шаблон проектирования, который использует отношение "один ко многим". В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.

Данный паттерн еще называют Publisher-Subscriber (издатель-подписчик), поскольку отношения издателя и подписчиков характеризуют действие данного паттерна: подписчики подписываются email-рассылку определенного сайта. Сайт-издатель с помощью email-рассылки уведомляет всех подписчиков о изменениях. А подписчики получают изменения и производят определенные действия: могут зайти на сайт, могут проигнорировать уведомления и т.д.

Когда использовать паттерн Наблюдатель?

- Когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях
- Когда общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главным, другая получает сообщения и реагирует на них. Отделение логики обеих сторон позволяет их рассматривать независимо и использовать отдельно друга от друга.
- Когда существует один объект, рассылающий сообщения, и множество подписчиков, которые получают сообщения. При этом точное число подписчиков заранее неизвестно и процессе работы программы может изменяться.

С помощью диаграмм UML данный шаблон можно выразить следующим образом:

Формальное определение паттерна на языке C# может выглядеть следующим образом:

```
1       interface IObservable
2       {
3           void AddObserver(IObserver o);
4           void RemoveObserver(IObserver o);
5           void NotifyObservers();
6       }
7       class ConcreteObservable : IObservable
8       {
9           private List<IObserver> observers;
10          public ConcreteObservable()
11          {
```

```
            observers = new List<IObserver>();
        }
        public void AddObserver(IObserver o)
        {
            observers.Add(o);
        }

        public void RemoveObserver(IObserver o)
        {
            observers.Remove(o);
        }

        public void NotifyObservers()
        {
            foreach (IObserver observer in observers)
                observer.Update();
        }
    }

interface IObserver
{
void Update();
    }
class ConcreteObserver :IObserver
{
public void Update()
{
}
    }
```

Участники

IObservable: представляет наблюдаемый объект. Определяет три метода: AddObserver() (для добавления наблюдателя), RemoveObserver() (удаление набюдателя) и NotifyObservers() (уведомление наблюдателей)

ConcreteObservable: конкретная реализация интерфейса IObservable. Определяет коллекцию объектов наблюдателей.

IObserver: представляет наблюдателя, который подписывается на все уведомления наблюдаемого объекта. Определяет метод Update(), который вызывается наблюдаемым объектом для уведомления наблюдателя.

ConcreteObserver: конкретная реализация интерфейса IObserver.

При этом наблюдаемому объекту не надо ничего знать о наблюдателе кроме того, что тот реализует метод Update(). С помощью отношения агрегации реализуется слабосвязанность обоих компонентов. Изменения в наблюдаемом объекте не виляют на наблюдателя и наоборот.

В определенный момент наблюдатель может прекратить наблюдение. И после этого оба объекта - наблюдатель и наблюдаемый могут продолжать существовать в системе независимо друг от друга.

Рассмотрим реальный пример применения шаблона. Допустим, у нас есть биржа, где проходят торги, и есть брокеры и банки, которые следят за поступающей информацией и в зависимости от поступившей информации производят определенные действия:

```
class Program
{
static void Main(string[] args)
```

```
4           {
5               Stock stock = new Stock();
6               Bank bank = new Bank("ЮнитБанк", stock);
7               Broker broker = new Broker("Иван Иваныч", stock);
8               // имитация торгов
9               stock.Market();
10              // брокер прекращает наблюдать за торгами
11              broker.StopTrade();
12              // имитация торгов
13              stock.Market();
14
15              Console.Read();
16          }
17      }
18
19      interface IObserver
20      {
21          void Update(Object ob);
22      }
23
24      interface IObservable
25      {
26          void      RegisterObserver(IObserver
27          o); void RemoveObserver(IObserver
28          o); void NotifyObservers();
29      }
30
31      class Stock : IObservable
32      {
33          StockInfo sInfo; // информация о торгах
34
35          List<IObserver> observers;
36          public Stock()
37          {
38              observers = new List<IObserver>();
39              sInfo= new StockInfo();
40          }
41          public void RegisterObserver(IObserver o)
42          {
43              observers.Add(o);
44          }
45
46          public void RemoveObserver(IObserver o)
47          {
48              observers.Remove(o);
49          }
50
51          public void NotifyObservers()
52          {
53              foreach(IObserver o in observers)
54              {
55                  o.Update(sInfo);
```

```
56              }
57          }
58
59          public void Market()
60          {
61              Random rnd = new Random();
62              sInfo.USD = rnd.Next(20, 40);
63              sInfo.Euro = rnd.Next(30, 50);
64              NotifyObservers();
65          }
66      }
67
68  class StockInfo
69  {
70  public int USD { get; set; }
71  public int Euro { get; set; }
72      }
73
74  class Broker : IObserver
75  {
76  public string Name { get; set; }
77  IObservable stock;
78  public Broker(string name, IObservable obs)
79  {
80  this.Name = name;
81  stock = obs;
82  stock.RegisterObserver(this);
83          }
84  public void Update(object ob)
85  {
86  StockInfo sInfo = (StockInfo)ob; 87
88  if(sInfo.USD>30)
89  Console.WriteLine("Брокер {0} продает доллары;  Курс доллара: {1}", this.Name, sInfo
90              else
91                  Console.WriteLine("Брокер {0} покупает доллары;  Курс доллара: {1}", this.Name, sIn
92          }
93          public void StopTrade()
94          {
95              stock.RemoveObserver(this);
96              stock=null;
97          }
98      }
99
100 class Bank : IObserver
101 {
102 public string Name { get; set; }
103 IObservable stock;
104 public Bank(string name, IObservable obs)
105 {
106 this.Name = name;
107 stock = obs;
```

```
    108 stock.RegisterObserver(this);
109             }
    110 public void Update(object ob)
    111 {
    112 StockInfo sInfo = (StockInfo)ob; 113
    114 if (sInfo.Euro > 40)
    115 Console.WriteLine("Банк {0} продает евро;  Курс евро: {1}", this.Name, sInfo.Euro);
    116 else
    117 Console.WriteLine("Банк {0} покупает евро;  Курс евро: {1}", this.Name, sInfo.Euro);
118             }
119         }
```

Итак, здесь наблюдаемый объект представлен интерфейсом IObservable, а наблюдатель - интерфейсом IObserver. Реализацией интерфейса IObservable является класс Stock, который символизирует валютную биржу. В этом классе определен метод Market(), который имитирует торги и инкапсулирует всю информацию о валютных курсах в объекте StockInfo. После проведения торгов производится уведомление всех наблюдателей.

Реализациями интерфейса IObserver являются классы Broker, представляющий брокера, и Bank, представляющий банк. При этом метод Update() интерфейса IObserver принимает в качестве параметра некоторый объект. Реализация этого метода подразумевает получение через данный параметр объекта StockInfo с текущей информацией о торгах и произведение некоторых действий: покупка или продажа долларов и евро. Дело в том, что часто необходимо информировать наблюдателя об изменении состояния наблюдаемого объекта. В данном случае состояние заключено в объекте StockInfo. И одним из вариантом информирования наблюдателя о состоянии является push-модель, при которой наблюдаемый объект передает (иначе говоря толкает - push) данные о своем состоянии, то есть передаем в виде параметра метода Update().

Альтернативой push-модели является pull-модель, когда наблюдатель вытягивает (pull) из наблюдаемого объекта данные о состоянии с помощью дополнительных методов.

Также в классе брокера определен дополнительный метод StopTrade(), с помощью которого брокер может отписаться от уведомлений биржи и перестать быть наблюдателем.

Практическая работа №19 Разработка приложения с использованием текстовых компонентов

Создание простого консольного приложения C# в Visual Studio

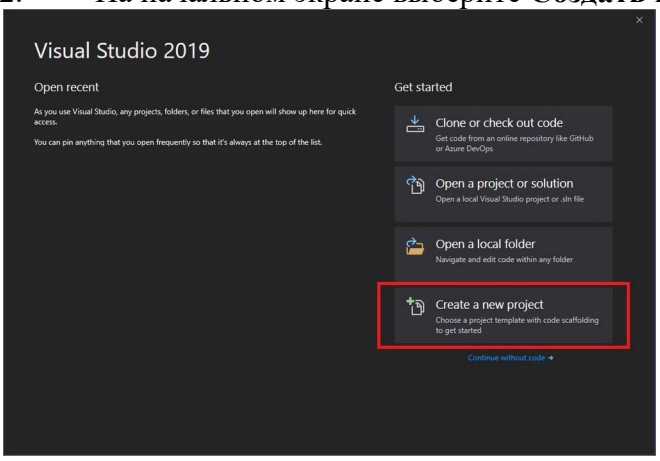
В этом учебнике по C# вы создадите и запустите консольное приложение с помощью Visual Studio, а также ознакомитесь с некоторыми возможностями интегрированной среды разработки (IDE) Visual Studio.

Установите Visual Studio бесплатно со страницы скачиваемых материалов Visual Studio, если еще не сделали этого.

Создание проекта

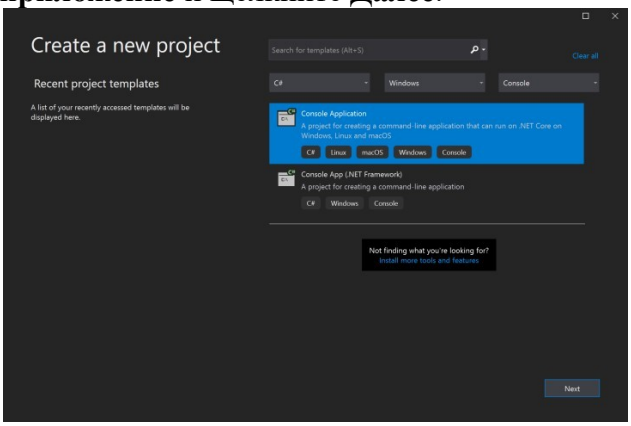
Для начала мы создадим проект приложения C#. Для этого типа проекта уже имеются все нужные файлы шаблонов, что избавляет вас от лишней работы.

1. Запустите Visual Studio 2019.
2. На начальном экране выберите **Создать проект**.



3. В окне **Создание проекта** выберите **C#** в списке языков. Затем выберите **Windows** в списке платформ и **Консоль** в списке типов проектов.

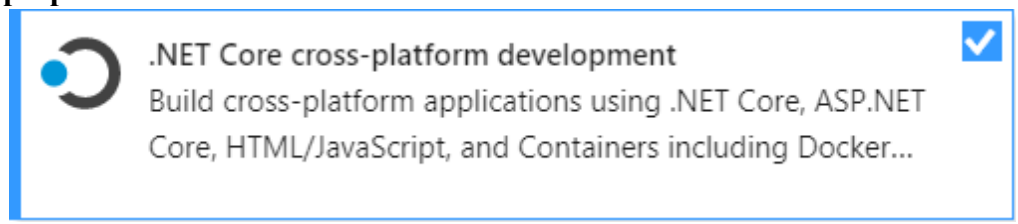
Применив фильтры по языку, платформе и типу проекта, выберите шаблон **Консольное приложение** и щелкните **Далее**.



Примечание

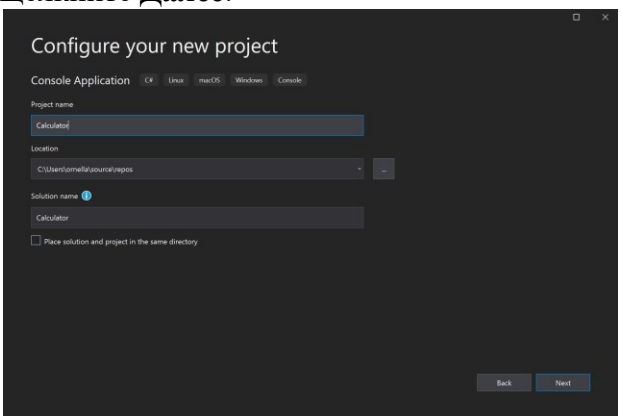
Если шаблон **Консольное приложение** отсутствует, его можно установить в окне **Создание проекта**. В сообщении **Не нашли то, что искали?** выберите ссылку **Установка других средств и компонентов**.

После этого в Visual Studio Installer выберите рабочую нагрузку **Кроссплатформенная разработка .NET Core**.

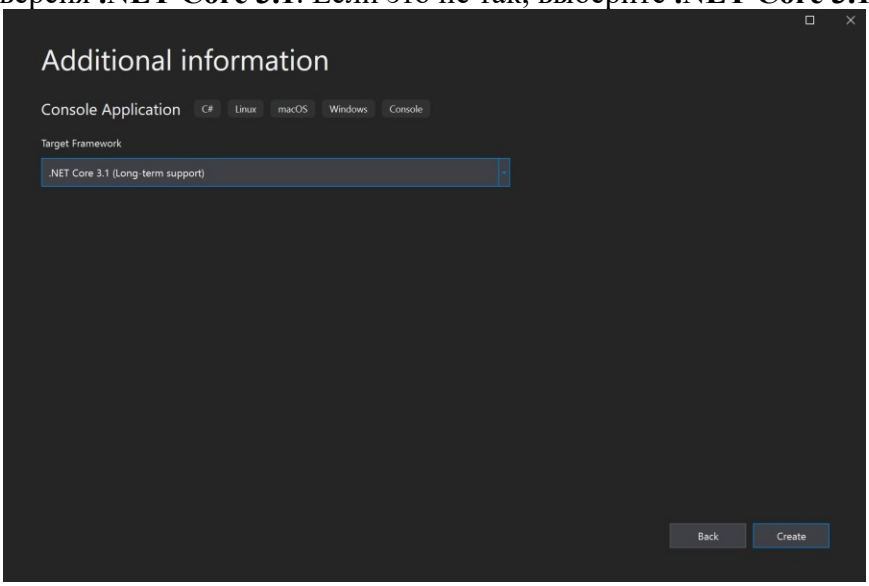


Затем нажмите кнопку **Изменить** в Visual Studio Installer. Вам может быть предложено сохранить результаты работы; в таком случае сделайте это. Выберите **Продолжить**, чтобы установить рабочую нагрузку. После этого вернитесь к шагу 2 в процедуре **Создание проекта**.

4. В поле **Имя проекта** окна **Настроить новый проект** введите *Calculator*. Затем щелкните **Далее**.



5. В окне **Дополнительные сведения** для целевой платформы должна быть указана версия **.NET Core 3.1**. Если это не так, выберите **.NET Core 3.1**. Затем нажмите **Создать**.



Visual Studio открывает новый проект, включающий код по умолчанию "Hello World".

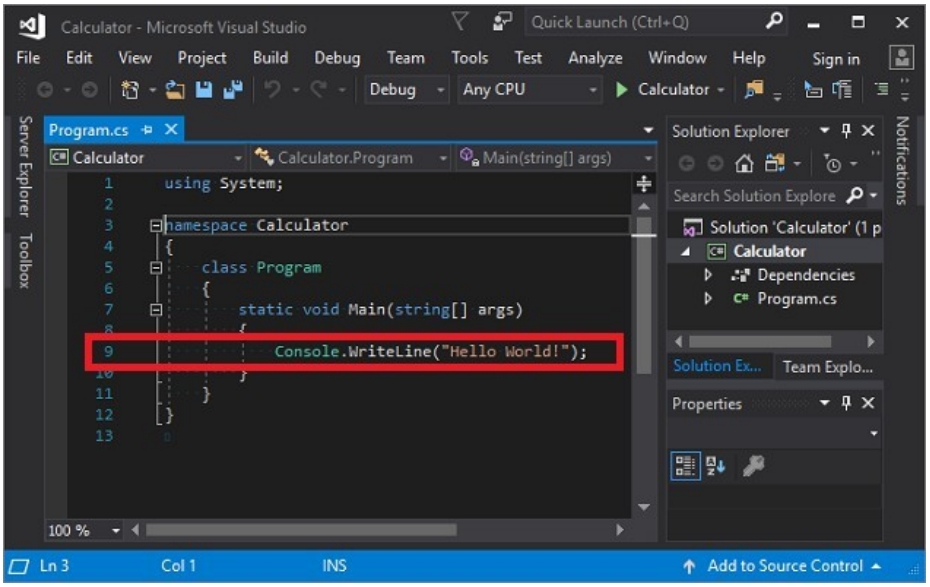
Создание приложения

Во-первых, мы рассмотрим некоторые базовые расчеты для целых чисел в C#. Затем мы добавим код для создания простого калькулятора. После этого нам предстоит отладить приложение, чтобы найти и исправить ошибки. И, наконец, мы оптимизируем код для повышения эффективности.

Вычисления с целыми числами

Давайте начнем с базовых расчетов целых чисел в C#.

1. В редакторе кода удалите созданный по умолчанию код Hello, World!.



В частности, удалите строку с текстом: Console.WriteLine("Hello World!");.

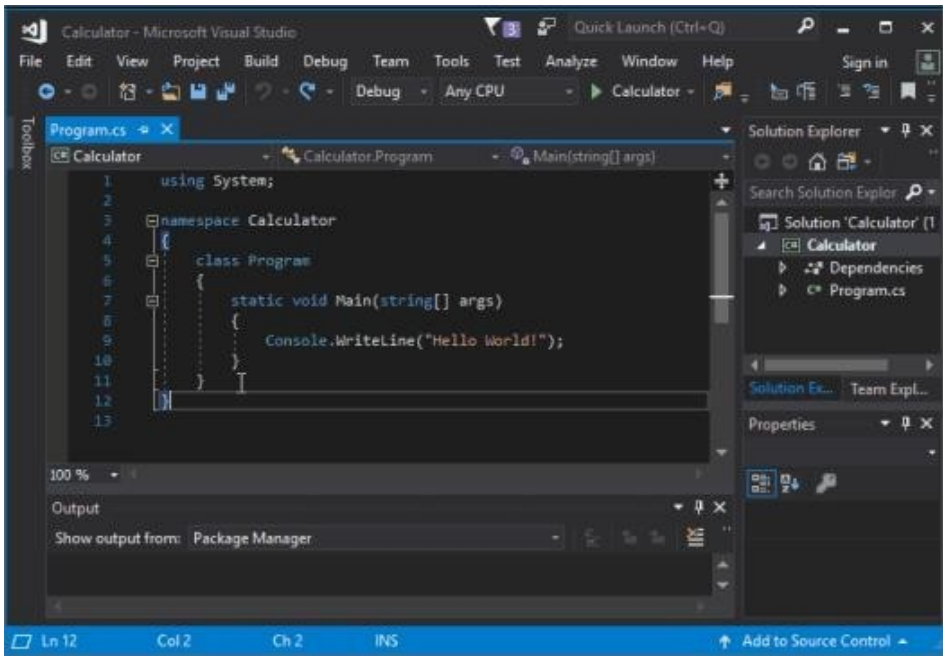
2. Вместо нее введите следующий код:

C#Копировать

```
int a = 42;
int b = 119;
int c = a + b;
Console.WriteLine(c);
Console.ReadKey();
```

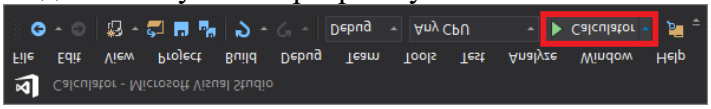
Обратите внимание на то, что при этом функция IntelliSense в Visual Studio предлагает возможность автовыполнения записи.

Примечание

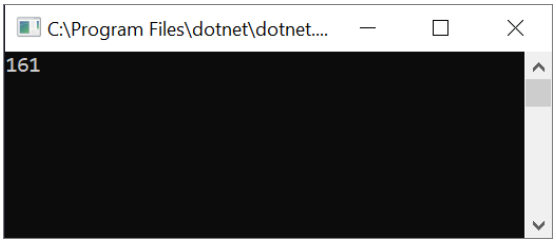


Следующая анимация не предназначена для дублирования предыдущего кода. Она предназначена только для того, чтобы продемонстрировать, как работает функция автозаполнения.

3. Нажмите зеленую кнопку **Пуск** или клавишу **F5** рядом с **калькулятором**, чтобы создать и запустить программу.



Откроется окно консоли с суммой 42 + 119, которая равна **161**.



4. **(Необязательно)** Можно изменить оператор, чтобы изменить результат. Например, можно изменить оператор + в строке кода int c = a + b; на - для вычитания, * для умножения или / для деления. Затем при запуске программы результат также изменится.
5. Закройте окно консоли.

Добавление кода для создания калькулятора

Давайте продолжим, добавляя более сложный набор кода калькулятора в проект.

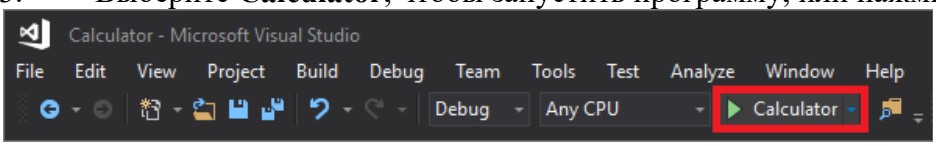
1. Удалите весь код, который отображается в редакторе кода.
2. Введите или вставьте в редактор кода следующий код:

C#Копировать

```
using System;

namespace Calculator
    {
      class Program
      {
        static void Main(string[] args)
        {
          // Declare variables and then initialize to zero.
          int num1 = 0; int num2 = 0;

          // Display title as the C# console calculator app.
          Console.WriteLine("Console Calculator in C#\r");
          Console.WriteLine("------------------------\n");

          // Ask the user to type the first number.

          Console.WriteLine("Type a number, and then press Enter");
          num1 = Convert.ToInt32(Console.ReadLine());

          // Ask the user to type the second number.
          Console.WriteLine("Type another number, and then press Enter");
          num2 = Convert.ToInt32(Console.ReadLine());

          // Ask the user to choose an option.
          Console.WriteLine("Choose an option from the following list:");
          Console.WriteLine("\ta - Add");
          Console.WriteLine("\ts - Subtract");
          Console.WriteLine("\tm - Multiply");
          Console.WriteLine("\td - Divide");
          Console.Write("Your option? ");

          // Use a switch statement to do the math.
          switch (Console.ReadLine())
          {
            case "a":
              Console.WriteLine($"Your result: {num1} + {num2} = " + (num1 + num2));
              break;
            case "s":
```

```
                Console.WriteLine($"Your result: {num1} - {num2} = " + (num1 - num2));
                break;
            case "m":
                Console.WriteLine($"Your result: {num1} * {num2} = " + (num1 * num2));
                break;
            case "d":
                Console.WriteLine($"Your result: {num1} / {num2} = " + (num1 / num2));
                break;
        }
        // Wait for the user to respond before closing.
        Console.Write("Press any key to close the Calculator console app...");
        Console.ReadKey();
    }
  }
}
```

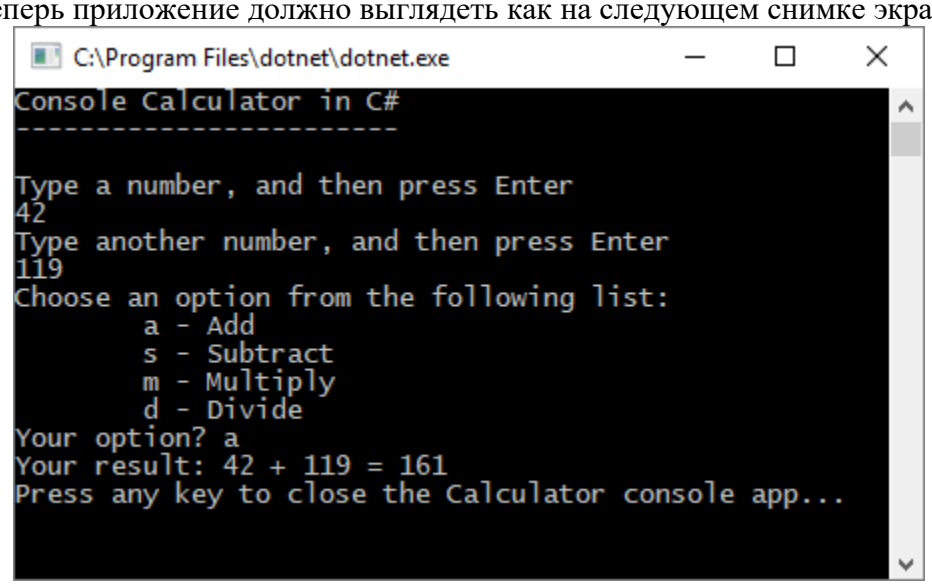
3. Выберите **Calculator**, чтобы запустить программу, или нажмите клавишу **F5**.



Откроется окно консоли.

4. Просмотрите приложение в окне консоли и сложите числа **42** и **119**, пользуясь предложенными подсказками.

Теперь приложение должно выглядеть как на следующем снимке экрана:



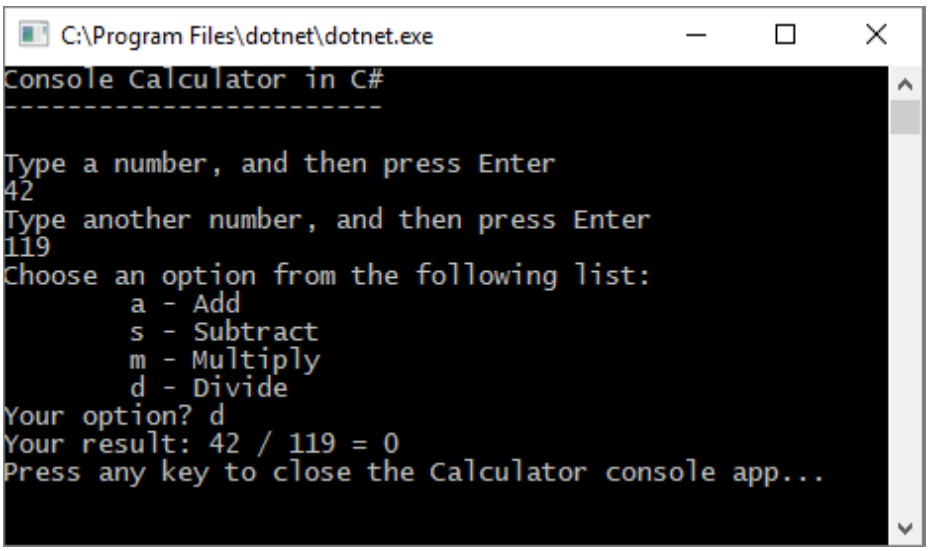
Добавление функциональных возможностей в калькулятор

Давайте изменим этот код, чтобы добавить функциональные возможности.

Обработка десятичных чисел

Пока наше приложение принимает и возвращает только целые числа. Вычисления можно сделать точнее, добавив код для обработки десятичных чисел.

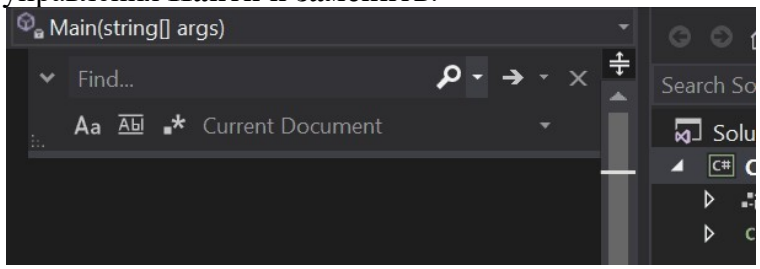
Как показано на следующем снимке экрана, при делении числа 42 на число 119 вы получите результат 0, что для нас недостаточно точно.



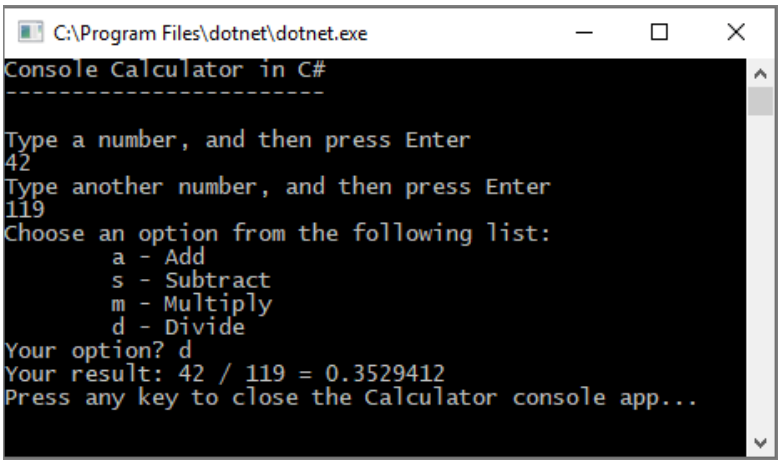
Давайте исправим код, чтобы он обрабатывал десятичные числа.

1. Нажмите клавиши **CTRL** + **H**, чтобы открыть элемент управления **Найти и заменить**.
2. Измените каждый экземпляр переменной int на float.

Переключите **Учитывать регистр** (**ALT**+**C**) и **Слово целиком** (**ALT**+**W**) в элементе управления **Найти и заменить**.

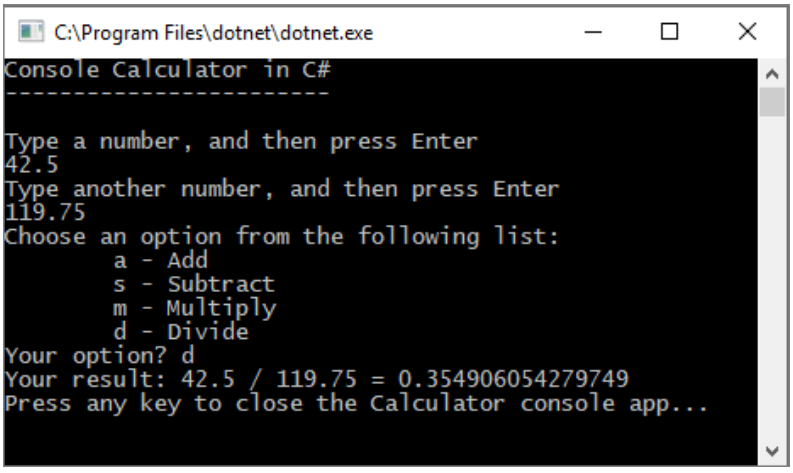


3. Еще раз запустите приложение калькулятора и разделите число **42** на число **119**. Обратите внимание, что теперь приложение возвращает не просто ноль, а десятичное число.



Но пока приложение только возвращает десятичные числа. Давайте изменим код так, чтобы приложение могло выполнять операции над десятичными числами.

1. Используйте элемент управления **Найти и заменить** (**CTRL** + **H**), чтобы изменить каждый экземпляр переменной float на double и каждый экземпляр метода Convert.ToInt32 на Convert.ToDouble.
2. Запустите приложение калькулятора и разделите число **42,5** на число **119,75**. Обратите внимание на то, что теперь приложение принимает и возвращает значения десятичные числа.



(Количество десятичных разрядов мы исправим с помощью инструкций по пересмотру кода.)

Отладка приложения

Мы уже улучшили наше простое приложение калькулятора, но пока оно не умеет обрабатывать исключения, включая ошибки во входных данных.

Например, при попытке разделить любое число на ноль или при вводе буквенного символа там, где приложение ожидает число (или наоборот), приложение может перестать работать, вернет ошибку или непредвиденный нечисловой результат.

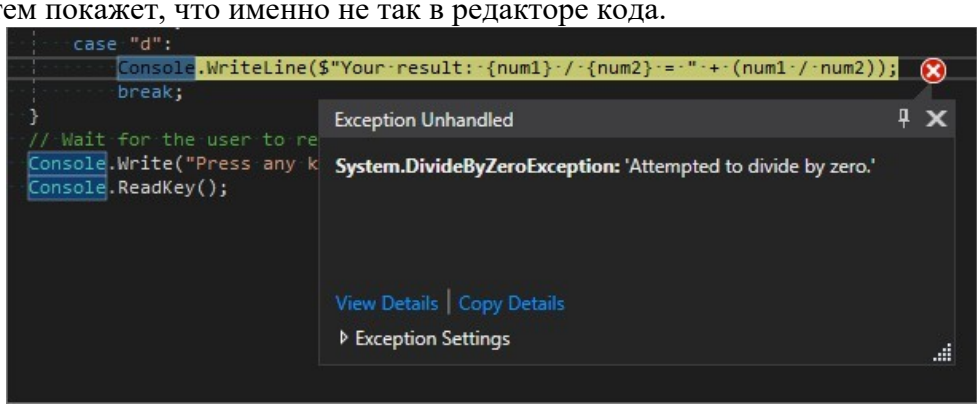
Давайте рассмотрим несколько типичных ошибок во входных данных, найдем их с помощью отладчика, если они там есть, и исправим код, чтобы устранить их.

Совет

Дополнительные сведения об отладчике и принципах его работы см. в разделе **Знакомство с отладчиком Visual Studio**.

Исправление ошибки деления на ноль

При попытке деления числа на ноль консольное приложение может перестать отвечать, а затем покажет, что именно не так в редакторе кода.



Примечание

Иногда приложение не зависает, а отладчик не отображает ошибку деления на ноль. Вместо этого приложение может вернуть непредвиденный нечисловой результат, например символ бесконечности. Приведенное ниже исправление кода по-прежнему применимо.

Давайте изменим код, чтобы он обрабатывал такую ошибку.

1. Удалите код между case "d": и комментарием с текстом // Wait for the user to respond before closing.
2. Замените его следующим кодом.

C#Копировать

```
// Ask the user to enter a non-zero divisor until they do so.
while (num2 == 0)
{
   Console.WriteLine("Enter a non-zero divisor: ");
   num2 = Convert.ToInt32(Console.ReadLine());
}
Console.WriteLine($"Your result: {num1} / {num2} = " + (num1 / num2));
break;
```

```
}
```

Когда вы добавите новый код, раздел с оператором switch будет выглядеть так, как показано на следующем снимке экрана:

```
// Use a switch statement to do the math
switch (Console.ReadLine())
{
    case "a":
        Console.WriteLine($"Your result: {num1} + {num2} = " + (num1 + num2));
        break;
    case "s":
        Console.WriteLine($"Your result: {num1} - {num2} = " + (num1 - num2));
        break;
    case "m":
        Console.WriteLine($"Your result: {num1} * {num2} = " + (num1 * num2));
        break;
    case "d":
        // Ask the user to enter a non-zero divisor until they do so
        while (num2 == 0)
        {
            Console.WriteLine("Enter a non-zero divisor: ");
            num2 = Convert.ToInt32(Console.ReadLine());
        }
        Console.WriteLine($"Your result: {num1} / {num2} = " + (num1 / num2));
        break;
}
// Wait for the user to respond before closing
Console.Write("Press any key to close the Calculator console app...");
Console.ReadKey();
```

Теперь при делении любого числа на ноль приложение предложит ввести другое число. Даже лучше: Оно будет снова и снова повторять этот запрос, пока не получит значение, отличающееся от нуля.

```
C:\Program Files\dotnet\dotnet.exe
Console Calculator in C#
------------------------

Type a number, and then press Enter
42
Type another number, and then press Enter
0
Choose an option from the following list:
        a - Add
        s - Subtract
        m - Multiply
        d - Divide
Your option? d
Enter a non-zero divisor:
0
Enter a non-zero divisor:
0
Enter a non-zero divisor:
2
Your result: 42 / 2 = 21
Press any key to close the Calculator console app...
```

Исправление ошибки формата

Если вы введете буквенный символ там, где приложение ожидает цифру (или наоборот), приложение консоли перестает работать. Visual Studio отображает причину проблемы в редакторе кода.

```
// Ask the user to type the first number
Console.WriteLine("Type a number, and then press Enter");
num1 = Convert.ToInt32(Console.ReadLine());

// Ask the user to type the second number
Console.WriteLine("Type another number, and
num2 = Convert.ToInt32(Console.ReadLine());

// Ask the user to choose an option
Console.WriteLine("Choose an option from the
Console.WriteLine("\ta - Add");
Console.WriteLine("\ts - Subtract");
Console.WriteLine("\tm - Multiply");
Console.WriteLine("\td - Divide");
Console.Write("Your option? ");
```

```
Exception Unhandled
System.FormatException: 'Input string was not in a correct format
View Details | Copy Details
Exception Settings
```

Чтобы устранить эту ошибку, мы выполним рефакторинг введенного ранее кода.

7. Пересмотр кода

Чтобы не делегировать всю обработку кода классу program, мы разделим приложение на два класса: Calculator и Program.

Класс Calculator выполняет основную часть работы для вычислений, а класс Program отвечает за пользовательский интерфейс и перехват ошибок.

Итак, начнем.

1. Удалите все в пространстве имен Calculator между открывающей и закрывающей фигурными скобками:

C#Копировать

```
using System;

namespace Calculator
    {

    }
```

2. Теперь добавьте новый класс Calculator со следующим содержимым:

C#Копировать

```
class
Calculator
    {
       public static double DoOperation(double num1, double num2, string op)
       {
      double result = double.NaN; // Default value is "not-a-number" which we use if an operation, such as division, could result in an error.

          // Use a switch statement to do the math.
          switch (op)
          {
             case "a":
                result = num1 + num2;
                break;
             case "s":
                result = num1 - num2;
                break;
             case "m":
                result = num1 * num2;
                break;
             case "d":
                // Ask the user to enter a non-zero divisor.
                if (num2 != 0)
                {
                   result = num1 / num2;
                }
                break;
             // Return text for an incorrect option entry.
             default:
                break;
          }
          return result;
       }
    }
```

3. Затем добавьте новый класс Program со следующим содержимым:

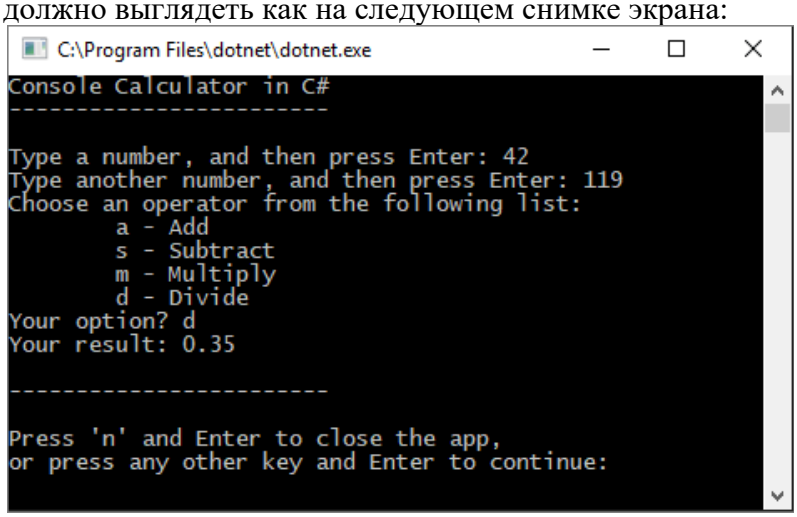
C#Копировать

```
class
Program
    {
```

```
static void Main(string[] args)
{
   bool endApp = false;
   // Display title as the C# console calculator app.
   Console.WriteLine("Console Calculator in C#\r");
   Console.WriteLine("------------------------\n");

   while (!endApp)
   {
      // Declare variables and set to empty.
      string numInput1 = "";
      string numInput2 = "";
      double result = 0;

      // Ask the user to type the first number.
      Console.Write("Type a number, and then press Enter: ");
      numInput1 = Console.ReadLine();

      double cleanNum1 = 0;
      while (!double.TryParse(numInput1, out cleanNum1))
      {
         Console.Write("This is not valid input. Please enter an integer value: ");
         numInput1 = Console.ReadLine();
      }

      // Ask the user to type the second number.
      Console.Write("Type another number, and then press Enter: ");
      numInput2 = Console.ReadLine();

      double cleanNum2 = 0;
      while (!double.TryParse(numInput2, out cleanNum2))
      {
         Console.Write("This is not valid input. Please enter an integer value: ");
         numInput2 = Console.ReadLine();
      }
      // Ask the user to choose an operator.
      Console.WriteLine("Choose an operator from the following list:");
      Console.WriteLine("\ta - Add");
      Console.WriteLine("\ts - Subtract");
      Console.WriteLine("\tm - Multiply");
      Console.WriteLine("\td - Divide");
      Console.Write("Your option? ");

      string op = Console.ReadLine();

      try
      {
         result = Calculator.DoOperation(cleanNum1, cleanNum2, op);
         if (double.IsNaN(result))
         {
            Console.WriteLine("This operation will result in a mathematical error.\n");
         }
         else Console.WriteLine("Your result: {0:0.##}\n", result);
      }
      catch (Exception e)
```

```
            {
                Console.WriteLine("Oh no! An exception occurred trying to do the math.\n - Details: "
+ e.Message);
            }

            Console.WriteLine("-----------------------\n");

            // Wait for the user to respond before closing.
        Console.Write("Press 'n' and Enter to close the app, or press any other key and Enter to
continue: ");
            if (Console.ReadLine() == "n") endApp = true;

            Console.WriteLine("\n"); // Friendly linespacing.
        }
        return;
    }
}
```

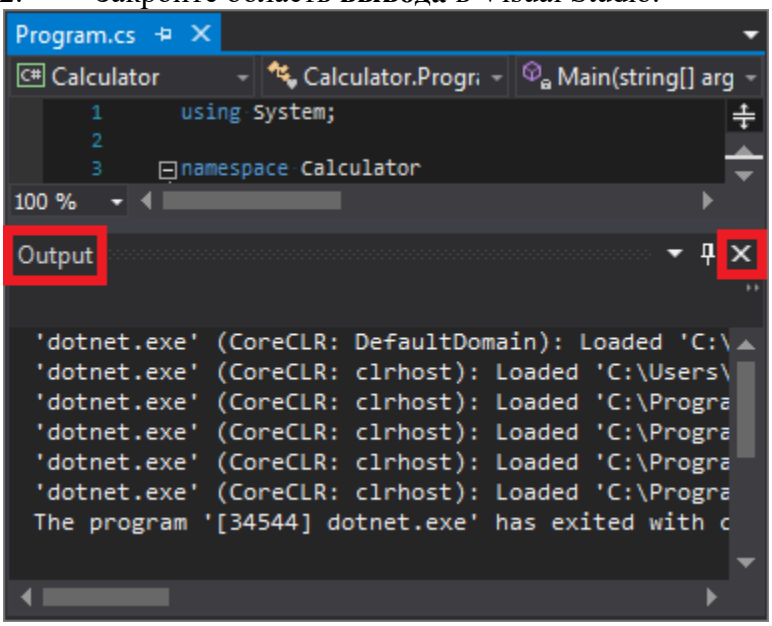
4. Выберите **Calculator**, чтобы запустить программу, или нажмите клавишу **F5**.
5. Разделите число **42** на число **119**, следуя подсказкам на экране. Теперь приложение должно выглядеть как на следующем снимке экрана:



Обратите внимание на то, что вы можете ввести несколько выражений, не закрывая консольное приложение. Также мы сократили количество десятичных разрядов в результате.

Закрытие приложения

1. Закройте приложение калькулятора, если оно еще открыто.
2. Закройте область **вывода** в Visual Studio.



3. В Visual Studio нажмите клавиши **CTRL**+**S**, чтобы сохранить приложение.

4. Закройте Visual Studio.

Полный код

В этом руководстве мы внесли много изменений в приложение "Калькулятор". Теперь оно более эффективно использует вычислительные ресурсы и обрабатывает большинство ошибок во входных данных.

Ниже мы собрали в один блок весь код:

C#Копировать

```
using System;

namespace Calculator
  {
    class Calculator
    {
      public static double DoOperation(double num1, double num2, string op)
      {
       double result = double.NaN; // Default value is "not-a-number" which we use if an operation, such as division, could result in an error.

        // Use a switch statement to do the math.
        switch (op)
        {
          case "a":
            result = num1 + num2;
            break;
          case "s":
            result = num1 - num2;
            break;
          case "m":
            result = num1 * num2;
            break;
          case "d":
            // Ask the user to enter a non-zero divisor.
            if (num2 != 0)
            {
              result = num1 / num2;
            }
            break;
          // Return text for an incorrect option entry.
          default:
            break;
        }
        return result;
      }
    }

    class Program
    {
      static void Main(string[] args)
      {
        bool endApp = false;
        // Display title as the C# console calculator app.
        Console.WriteLine("Console Calculator in C#\r");
        Console.WriteLine("------------------------\n");
```

```
while (!endApp)
{
  // Declare variables and set to empty.
  string numInput1 = "";
  string numInput2 = "";
  double result = 0;

  // Ask the user to type the first number.
  Console.Write("Type a number, and then press Enter: ");
  numInput1 = Console.ReadLine();
  double cleanNum1 = 0;
  while (!double.TryParse(numInput1, out cleanNum1))
  {
    Console.Write("This is not valid input. Please enter an integer value: ");
    numInput1 = Console.ReadLine();
  }

  // Ask the user to type the second number.
  Console.Write("Type another number, and then press Enter: ");
  numInput2 = Console.ReadLine();

  double cleanNum2 = 0;
  while (!double.TryParse(numInput2, out cleanNum2))
  {
    Console.Write("This is not valid input. Please enter an integer value: ");
    numInput2 = Console.ReadLine();
  }

  // Ask the user to choose an operator.
  Console.WriteLine("Choose an operator from the following list:");
  Console.WriteLine("\ta - Add");
  Console.WriteLine("\ts - Subtract");
  Console.WriteLine("\tm - Multiply");
  Console.WriteLine("\td - Divide");
  Console.Write("Your option? ");

  string op = Console.ReadLine();

  try
  {
    result = Calculator.DoOperation(cleanNum1, cleanNum2, op);
    if (double.IsNaN(result))
    {
      Console.WriteLine("This operation will result in a mathematical error.\n");
    }
    else Console.WriteLine("Your result: {0:0.##}\n", result);
  }
  catch (Exception e)
  {
  Console.WriteLine("Oh no! An exception occurred trying to do the math.\n - Details: " + e.Message);
  }

  Console.WriteLine("-----------------------\n");
```

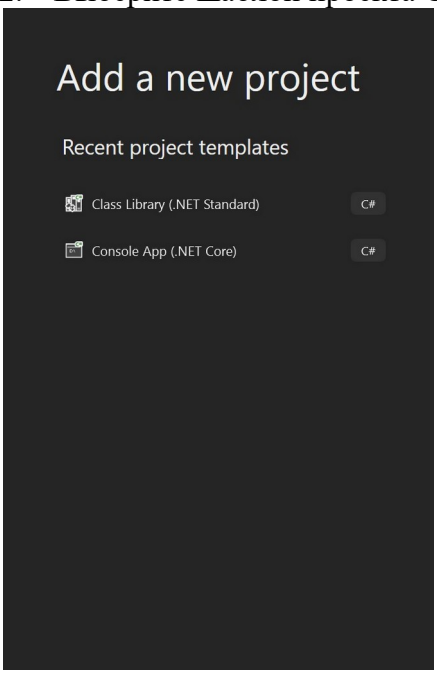
```
            // Wait for the user to respond before closing.
            Console.Write("Press 'n' and Enter to close the app, or press any other key and Enter to continue: ");
            if (Console.ReadLine() == "n") endApp = true;

            Console.WriteLine("\n"); // Friendly linespacing.}

        return;
      }
    }
  }
```

Практическая работа №20 Разработка приложения с несколькими формами.

Добавление нового проекта

Реальный код связан со множеством проектов, объединенных в решение. Давайте добавим еще один проект в приложение калькулятора. Это будет библиотека классов, предоставляющая некоторые функции калькулятора.

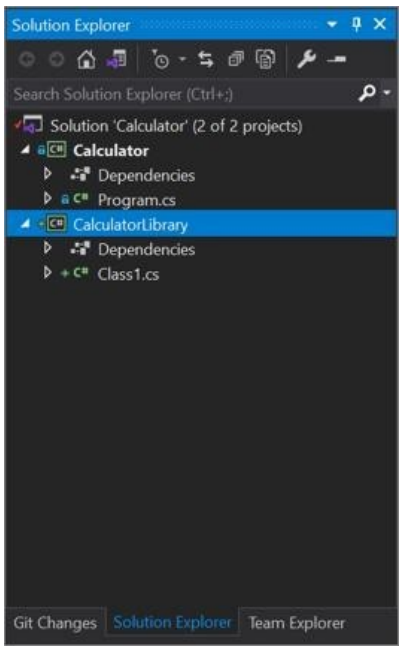
1. В Visual Studio можно использовать команду меню верхнего уровня **Файл** > **Добавить** > **Новый проект**, чтобы добавить новый проект, но можно также щелкнуть правой кнопкой мыши имя существующего проекта ("узел проекта") и открыть контекстное меню проекта. Это контекстное меню предлагает различные варианты добавления функциональных возможностей в проекты. Щелкните правой кнопкой мыши узел проекта в **обозревателе решений** и последовательно выберите **Добавить** > **Новый проект**.
2. Выберите шаблон проекта C# **Библиотека классов (.NET Standard)** .



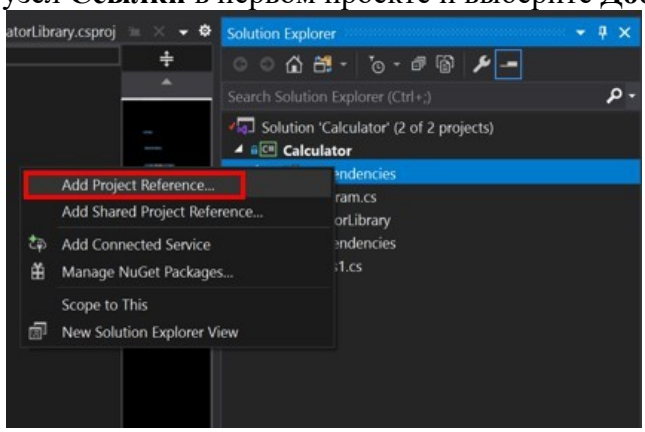
Введите имя проекта **CalculatorLibrary** и щелкните **Создать**. Visual Studio создаст новый проект и добавит его в решение.

3. Вместо *Class1.cs* переименуйте файл в **CalculatorLibrary.cs**. Можно щелкнуть имя в **обозреватель решений**, чтобы указать другое имя, или щелкнуть правой кнопкой мыши и выбрать **Переименовать** или нажать клавишу **F2**.

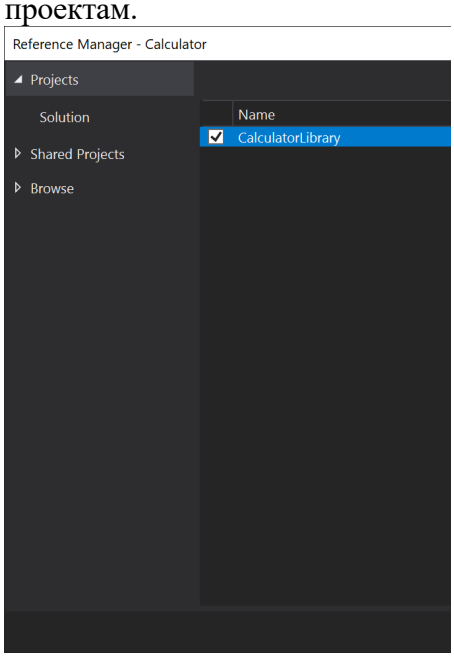
Возможно, вам будет предложено переименовать все ссылки на Class1 в файле. Вы можете выбрать любой ответ, так как вы замените код на следующем шаге.



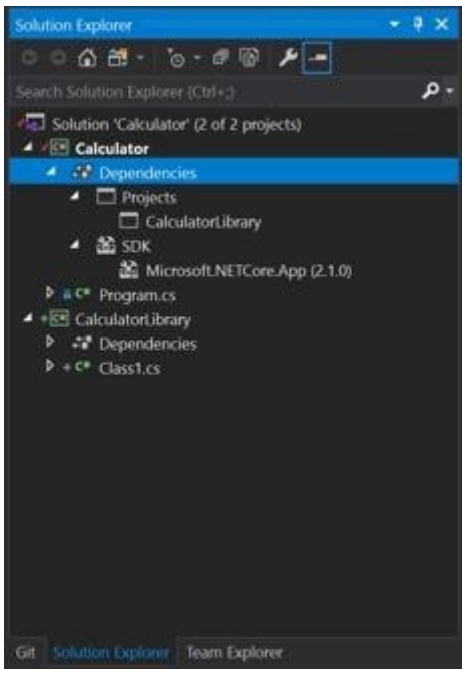
4. Теперь необходимо добавить ссылку на проект, чтобы первый проект мог использовать API, предоставляемые новой библиотекой классов. Щелкните правой кнопкой мыши узел **Ссылки** в первом проекте и выберите **Добавить ссылку на проект**.



Откроется диалоговое окно **Диспетчер ссылок**. Это диалоговое окно позволяет добавлять ссылки на другие проекты, а также сборки и библиотеки DLL COM, необходимые вашим проектам.



В диалоговом окне **Диспетчер ссылок** установите флажок для проекта **CalculatorLibrary** и выберите **ОК**. Ссылка на проект отображается в узле **Проекты** в **обозревателе решений**.



5. В файле *Program.cs* выберите класс Calculator и весь его код и нажмите **CTRL+X**, чтобы вырезать его из Program.cs. Затем в **CalculatorLibrary**, в файле *CalculatorLibrary.cs*, вставьте код в пространство имен CalculatorLibrary. Затем объявите класс калькулятора как public, чтобы предоставить его за пределами библиотеки. Теперь код в *CalculatorLibrary.cs* должен выглядеть следующим образом:

C#Копировать

```
using System;

 namespace CalculatorLibrary
 {
    public class Calculator
    {
       public static double DoOperation(double num1, double num2, string op)
       {
      double result = double.NaN; // Default value is "not-a-number" which we use if an operation, such as division, could result in an error.

            // Use a switch statement to do the math.
            switch (op)
            {
               case "a":
                  result = num1 + num2;
                  break;
               case "s":
                  result = num1 - num2;
                  break;
               case "m":
                  result = num1 * num2;
                  break;
               case "d":
                  // Ask the user to enter a non-zero divisor.
                  if (num2 != 0)
                  {
                     result = num1 / num2;
                  }
                  break;
```

```
            // Return text for an incorrect option entry.
            default:
               break;
         }
         return result;
      }
   }
}
```

6. Первый проект содержит ссылку, но вы увидите ошибку, так как вызов Calculator.DoOperation не разрешается. Это связано с тем, что CalculatorLibrary находится в другом пространстве имен, поэтому добавьте пространство имен CalculatorLibrary для полной ссылки.

C#Копировать

```
result = CalculatorLibrary.Calculator.DoOperation(cleanNum1, cleanNum2, op);
```

Попробуйте добавить директиву using в начало файла:

C#Копировать

```
using CalculatorLibrary;
```

Это изменение позволяет удалить пространство имен CalculatorLibrary из места вызова, но это приводит к неоднозначности. Является ли Calculator классом в CalculatorLibrary или Calculator является пространством имен? Чтобы устранить неоднозначность, переименуйте пространство имен CalculatorProgram.

C#Копировать

```
namespace CalculatorProgram
```

Ссылки на библиотеки .NET: запись в журнал

1. Предположим, что теперь нужно добавить журнал всех операций и записать его в текстовый файл. Класс .NET Trace предоставляет эти функции. (Это удобно и для основных методов отладки вывода.) Класс Trace находится в пространстве имен System.Diagnostics. Поскольку нам понадобятся классы System.IO, такие как StreamWriter, следует начать с добавления директив using.

C#Копировать

```
using System.IO;
using System.Diagnostics;
```

2. Просмотрев, как используется класс Trace, придерживайтесь ссылки на класс, связанный с файловым потоком. Это означает, что калькулятор будет работать лучше в качестве объекта, поэтому добавим конструктор.

C#Копировать

```
public Calculator()
      {
         StreamWriter logFile = File.CreateText("calculator.log");
         Trace.Listeners.Add(new TextWriterTraceListener(logFile));
         Trace.AutoFlush = true;
         Trace.WriteLine("Starting Calculator Log");
         Trace.WriteLine(String.Format("Started {0}", System.DateTime.Now.ToString()));
      }

   public double DoOperation(double num1, double num2, string op)
      {
```

3. И нам нужно изменить статический метод DoOperation на метод-член. Давайте также добавим выходные данные в каждый расчет для журнала, чтобы DoOperation выглядел следующим образом:

C#Копировать

```
public double DoOperation(double num1, double num2, string op)
   {
   double result = double.NaN; // Default value is "not-a-number" which we use if an operation, such as division, could result in an error.
```

```
        // Use a switch statement to do the math.
        switch (op)
        {
            case "a":
                result = num1 + num2;
                Trace.WriteLine(String.Format("{0} + {1} = {2}", num1, num2, result));
                break;
            case "s":
                result = num1 - num2;
                Trace.WriteLine(String.Format("{0} - {1} = {2}", num1, num2, result));
                break;
            case "m":
                result = num1 * num2;
                Trace.WriteLine(String.Format("{0} * {1} = {2}", num1, num2, result));
                break;
            case "d":
                // Ask the user to enter a non-zero divisor.
                if (num2 != 0)
                {
                    result = num1 / num2;
                    Trace.WriteLine(String.Format("{0} / {1} = {2}", num1, num2, result));
                }
                    break;
            // Return text for an incorrect option entry.
            default:
                break;
        }
        return result;
    }
```

4. Теперь вернемся к файлу Program.cs. Статический вызов помечен красной волнистой линией. Чтобы устранить эту проблему, создайте переменную calculator, добавив следующую строку непосредственно перед циклом while:

C#Копировать

```
Calculator calculator = new Calculator();
```

Измените сайт вызова для DoOperation следующим образом:

C#Копировать

```
result = calculator.DoOperation(cleanNum1, cleanNum2, op);
```

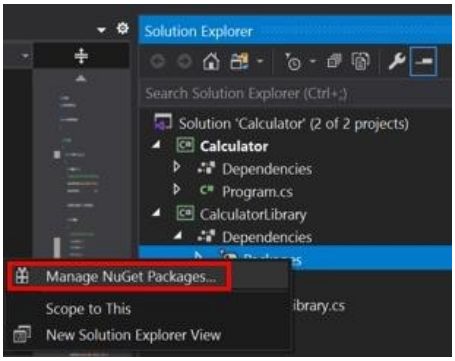
5. Запустите программу еще раз, а затем щелкните правой кнопкой мыши узел проекта, выберите **Открыть папку в проводнике файлов**, перейдите к выходной папке, например *bin/Debug/netcoreapp3.1*, и откройте файл *calculator.log*.

Выходные данныеКопировать

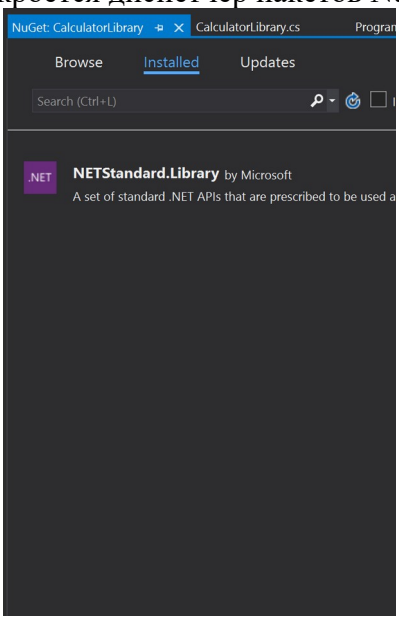
```
Starting Calculator Log
Started 7/9/2020 1:58:19 PM
1 + 2 = 3
3 * 3 = 9
```

Добавление пакета NuGet: запись в JSON-файл

1. Теперь предположим, что мы хотим вывести операции в формате JSON — популярном и переносимом формате для хранения данных объекта. Чтобы реализовать эту функциональность, необходимо сослаться на пакет NuGet Newtonsoft.Json. Пакеты NuGet являются основным средством распространения библиотек классов .NET. В **обозревателе решений** щелкните правой кнопкой мыши узел **Ссылки** для проекта CalculatorLibrary и выберите **Управление пакетами NuGet**.

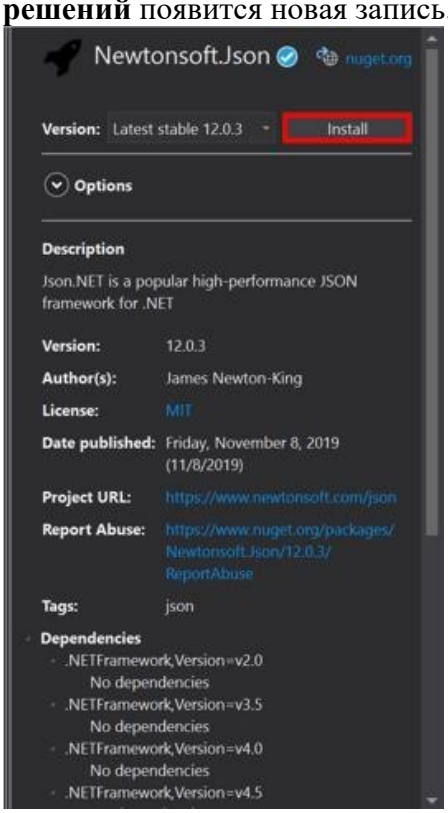


Откроется диспетчер пакетов NuGet.



2. Найдите пакет Newtonsoft.Json и нажмите **Установить**.

Пакет будет загружен и добавлен в проект, а в узле "Ссылки" в **обозревателе решений** появится новая запись.



3. Добавьте директиву using для System.IO и пакета Newtonsoft.Json в начало файла *CalculatorLibrary.cs*.

C#Копировать

```
using Newtonsoft.Json;
```

4. Теперь замените конструктор для калькулятора на следующий код и создайте объект члена JsonWriter:

C#Копировать

```
JsonWriter writer;

public Calculator()
{
   StreamWriter logFile = File.CreateText("calculatorlog.json");
   logFile.AutoFlush = true;
   writer = new JsonTextWriter(logFile);
   writer.Formatting = Formatting.Indented;
   writer.WriteStartObject();
   writer.WritePropertyName("Operations");
   writer.WriteStartArray();
}
```

5. Измените метод DoOperation, чтобы добавить код модуля записи JSON:

C#Копировать

```
public double DoOperation(double num1, double num2, string op)
{
double result = double.NaN; // Default value is "not-a-number" which we use if an operation, such as division, could result in an error.
   writer.WriteStartObject();
   writer.WritePropertyName("Operand1");
   writer.WriteValue(num1);
   writer.WritePropertyName("Operand2");
   writer.WriteValue(num2);
   writer.WritePropertyName("Operation");
   // Use a switch statement to do the math.
   switch (op)
   {
      case "a":
         result = num1 + num2;
         writer.WriteValue("Add");
         break;
      case "s":
         result = num1 - num2;
         writer.WriteValue("Subtract");
         break;
      case "m":
         result = num1 * num2;
         writer.WriteValue("Multiply");
         break;
      case "d":
         // Ask the user to enter a non-zero divisor.
         if (num2 != 0)
         {
            result = num1 / num2;
            writer.WriteValue("Divide");
         }
         break;
      // Return text for an incorrect option entry.
      default:
         break;
   }
```

```
        writer.WritePropertyName("Result");
        writer.WriteValue(result);
        writer.WriteEndObject();

        return result;
    }
```

6. Необходимо добавить метод для завершения синтаксиса JSON после ввода пользователем всех данных для операции.

C#Копировать

```
public void Finish()
{
    writer.WriteEndArray();
    writer.WriteEndObject();
    writer.Close();
}
```

7. В *Program.cs* добавьте вызов Finish в конце.

```
        // And call to close the JSON writer before return
        calculator.Finish();
        return;
    }
```

8. Выполните сборку приложения и запустите его. Попробуйте выполнить несколько операций и закройте приложение, используя команду "n". Теперь откройте файл calculatorlog.json, который будет содержать примерно следующее.

JSONКопировать

```
{
 "Operations": [
    {
    "Operand1": 2.0,
    "Operand2": 3.0,
    "Operation": "Add",
    "Result": 5.0
    },
    {
    "Operand1": 3.0,
    "Operand2":          4.0,
    "Operation":    "Multiply",
    "Result": 12.0
    }
 ]
}
```

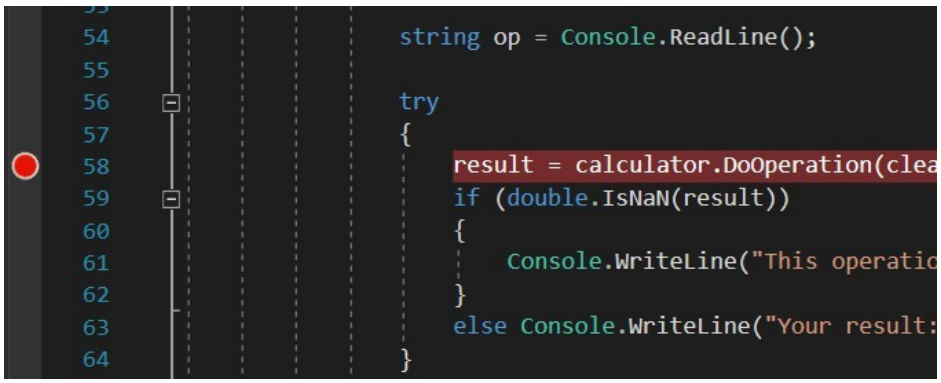
Отладка. Установка точки останова и попадание в нее

Отладчик Visual Studio — мощное средство для пошагового выполнения кода в поисках точки, в которой вы допустили ошибку при написании программы. Таким образом, вы получите возможность проанализировать код и внести в него необходимые исправления. Visual Studio позволяет вносить временные изменения, чтобы можно было продолжить выполнение программы.

1. В *Program.cs* щелкните поле слева от указанного ниже кода (или откройте контекстное меню и выберите пункт **Точка останова** > **Вставить точку останова**, либо нажмите клавишу **F9**):

C#Копировать

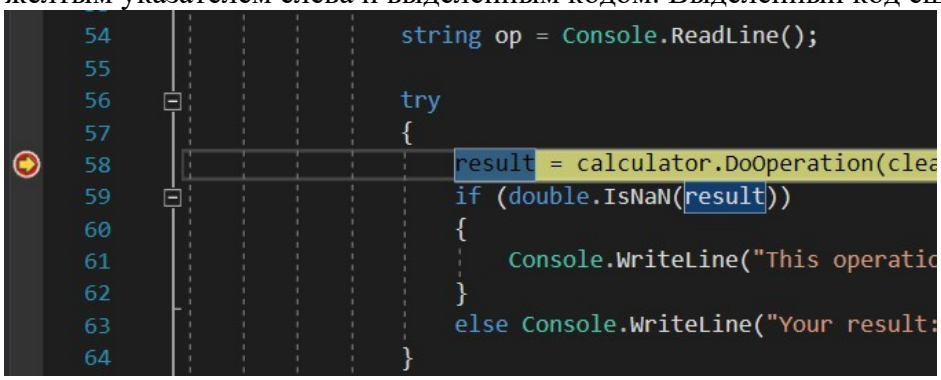
```
result = calculator.DoOperation(cleanNum1, cleanNum2, op);
```



Красный кружок, который отображается, указывает на точку останова. Для приостановки приложения и проверки кода можно использовать точки останова. Вы можете установить точку останова в любой исполняемой строке кода.

2. Выполните сборку и запустите приложение.
3. В выполняющемся приложении введите некоторые значения для вычисления:
   - Для первого числа введите **8**.
   - Для второго числа введите **0**.
   - В качестве оператора введите **d**.

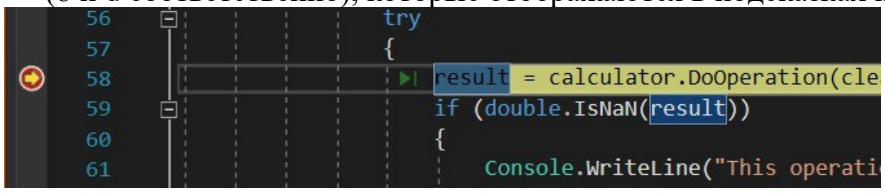
Приложение приостанавливается в созданной точке останова, которая обозначается желтым указателем слева и выделенным кодом. Выделенный код еще не выполнен.



Теперь, когда приложение приостановлено, вы можете проверить состояние приложения.

Отладка. Просмотр переменных

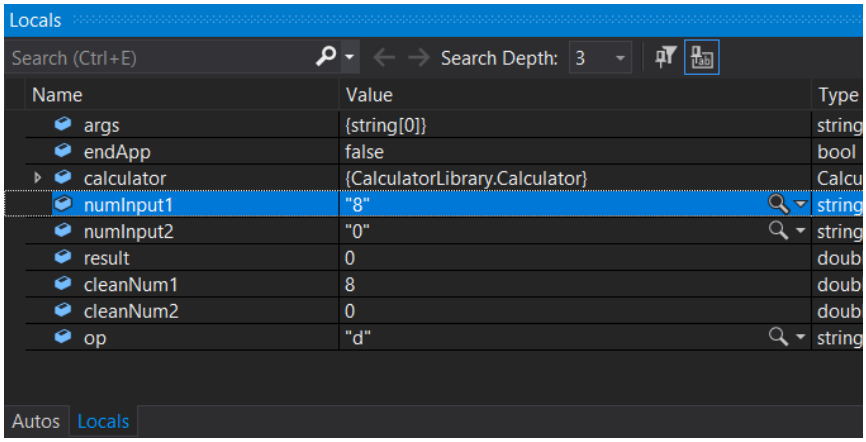
1. В выделенном коде наведите указатель мыши на переменные, такие как cleanNum1 и op. Вы увидите текущие значения этих переменных (8 и d соответственно), которые отображаются в подсказках по данным.



При отладке очень важно выполнить проверку переменных на содержание соответствующих значений для устранения проблем.

2. В нижней области просмотрите окно **Локальные**. (Если оно закрыто, выберите **Отладка** > **Окна** > **Локальные**, чтобы открыть его.)

В окне "Локальные" отображаются все переменные, находящиеся в области, а также их значения и типы.



3. Взгляните на окно **Видимые**.

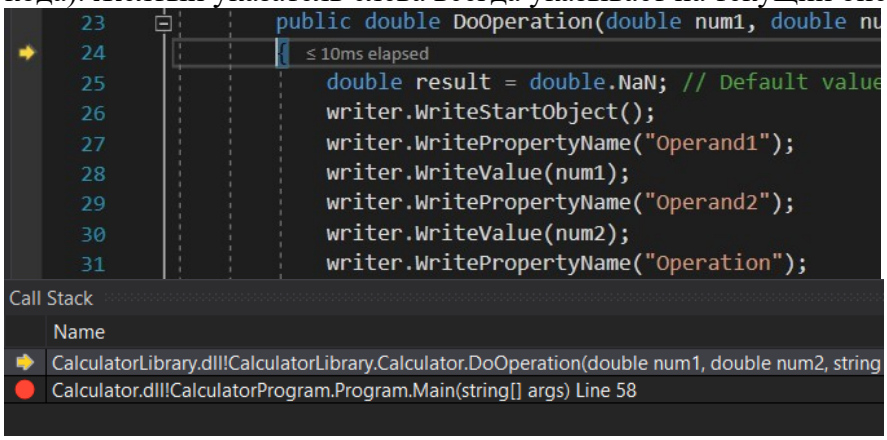
Окно "Видимые" аналогично окну **Локальные**, но отображает переменную непосредственно перед текущей строкой кода и после строки, в которой приостановлено приложение.

Далее можно поочередно выполнить код в отладчике по одной инструкции за раз. Это называется *пошаговым выполнением*.

Отладка. Пошаговое прохождение кода

1. Нажмите клавишу **F11** (или выберите **Отладка** > **Выполнять по шагам**).

С помощью команды "Выполнять по шагам" приложение выполняет текущий оператор и переходит к следующему исполняемому оператору (как правило, это следующая строка кода). Желтый указатель слева всегда указывает на текущий оператор.



Вы только что выполнили по шагам метод DoOperation в классе Calculator.

2. Чтобы получить иерархический обзор выполнения программы, просмотрите окно **Стек вызовов**. (Если оно закрыто, выберите **Отладка** > **Окна** > **Стек вызовов**.)

В этом представлении отображается текущий метод Calculator.DoOperation, обозначенный желтым указателем, а во второй строке показана функция, которая его вызвала из метода Main в *Program.cs*. В окне **Стек вызовов** показан порядок вызова методов и функций. Кроме того, оно предоставляет доступ ко многим функциям отладчика, таким как **Перейти к исходному коду**, из контекстного меню.

3. Нажмите клавишу **F10** (или выберите **Отладка** > **Шаг с обходом**) несколько раз, пока приложение не остановится в операторе switch.

C#Копировать

```
switch (op)
    {
```

Команда "Шаг с обходом" аналогична команде "Выполнять по шагам", за исключением того, что если текущий оператор вызывает функцию, то отладчик выполняет код в вызываемой функции и не приостанавливает выполнение до возврата функции. Команда "Шаг с обходом" — это более быстрый способ навигации по коду, если вас не интересует определенная функция.

4. Нажмите клавишу **F10** еще раз, чтобы приложение приостанавливалось на следующей строке кода.

C#Копировать

```
if (num2 != 0)
```

{

Этот код проверяет деление на нуль. Если приложение продолжает работать, оно вызовет общее исключение (ошибку). Предположим, вы считаете, что это ошибка, и хотите выполнить другое действие, например просмотреть фактическое возвращаемое значение в консоли. Один из вариантов — использовать функцию отладчика "Изменить и продолжить", чтобы внести изменения в код и продолжить отладку. Тем не менее мы покажем другой метод для временного изменения потока выполнения.

Отладка. Тестирование временного изменения

1. Выберите желтый указатель, который в данный момент приостановлен на операторе if (num2 != 0), и перетащите его в следующий оператор.

C#Копировать

result = num1 / num2;

При этом приложение полностью пропускает оператор if, чтобы вы могли увидеть, что происходит при делении на ноль.

2. Нажмите клавишу **F10**, чтобы выполнить строку кода.
3. Наведите указатель мыши на переменную result, вы увидите, что она сохраняет значение Infinity.

В C# Infinity является результатом деления на ноль.

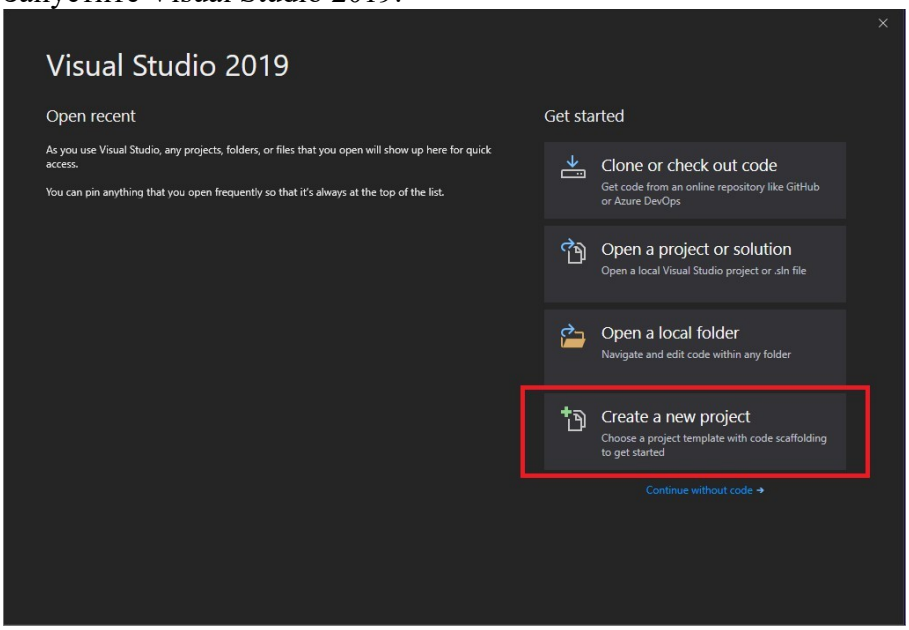
4. Нажмите клавишу **F5** (или выберите **Отладка** > **Продолжить отладку**).

Символ бесконечности отображается в консоли как результат математической операции.

5. Закройте приложение должным образом с помощью команды n.

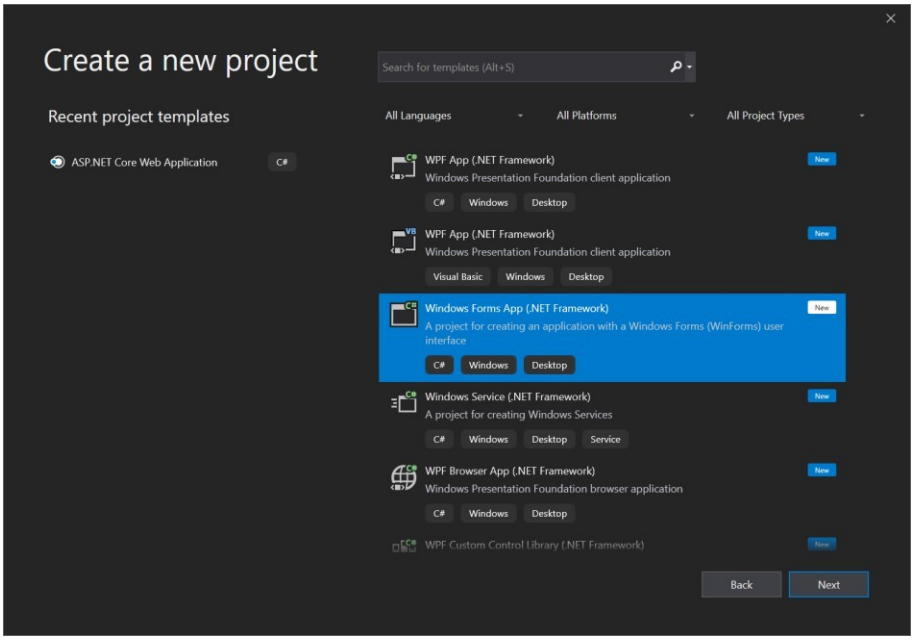
Практическая работа №21 Разработка приложения с не визуальными компонентами Сначала вы создадите проект приложения на C#. Для этого типа проекта уже имеются все нужные файлы шаблонов, что избавляет вас от лишней работы.

1. Запустите Visual Studio 2019.



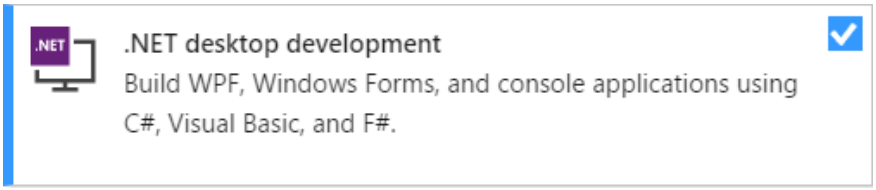
2.
3. На начальном экране выберите **Создать проект**.
4. В окне **Создать проект** выберите шаблон **Приложение Windows Forms (.NET Framework)** для C#.

(При желании вы можете уточнить условия поиска, чтобы быстро перейти к нужному шаблону. Например, введите *Приложение Windows Forms* в поле поиска. Затем выберите **C#** в списке языков и **Windows** в списке платформ.)



Примечание

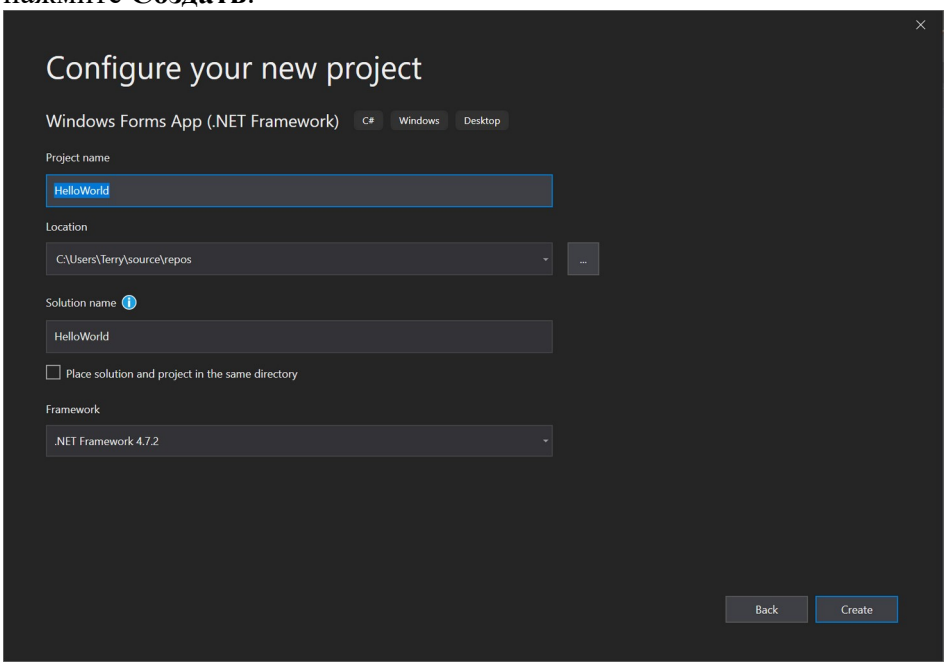
Если шаблон **Приложение Windows Forms (.NET Framework)** отсутствует, его можно установить из окна **Создание проекта**. В сообщении **Не нашли то, что искали?** выберите ссылку **Установка других средств и компонентов**.



После этого в Visual Studio Installer выберите рабочую нагрузку **Разработка классических приложений .NET**.

Затем нажмите кнопку **Изменить** в Visual Studio Installer. Вам может быть предложено сохранить результаты работы; в таком случае сделайте это. Выберите **Продолжить**, чтобы установить рабочую нагрузку. После этого вернитесь к шагу 2 в процедуре **Создание проекта**.

5. В поле **Имя проекта** окна **Настроить новый проект** введите *HelloWorld*. Затем нажмите **Создать**.



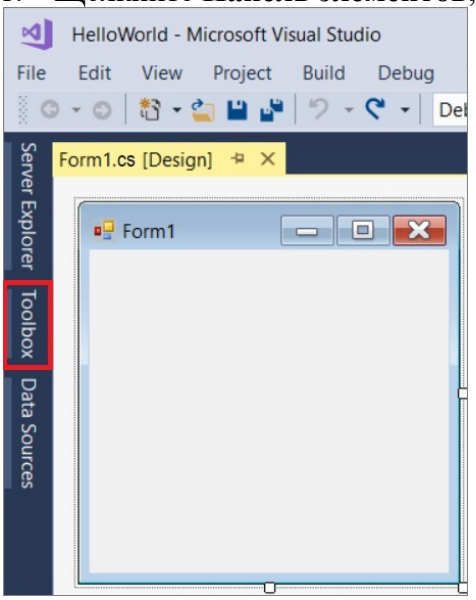
Новый проект открывается в Visual Studio.

Создание приложения

Когда вы выберете шаблон проекта C# и зададите имя файла, Visual Studio открывает форму. Форма является пользовательским интерфейсом Windows. Мы создадим приложение Hello World, добавив элементы управления на форму, а затем запустим его.

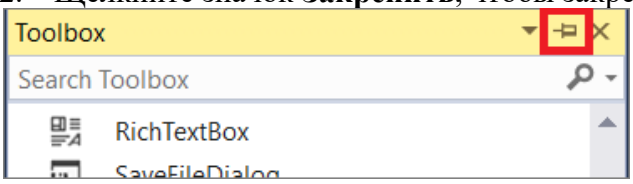
Добавление кнопки на форму

1. Щелкните **Панель элементов**, чтобы открыть всплывающее окно "Панель элементов".

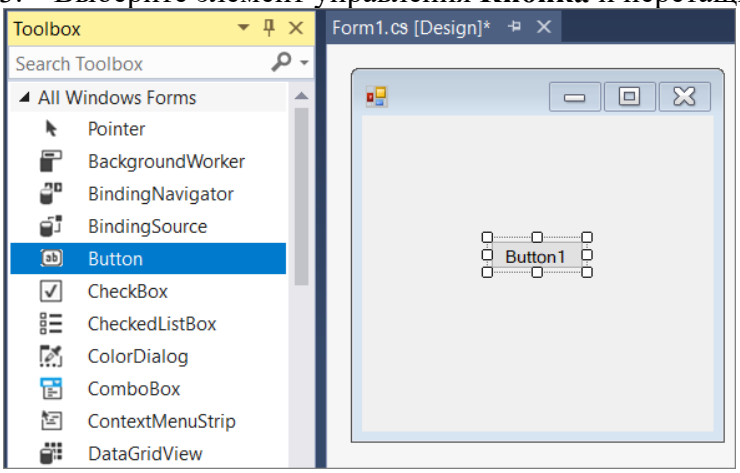


(Если параметр для всплывающего окна **Панель элементов** отсутствует, его можно открыть в строке меню. Для этого выберите **Вид** > **Панель элементов**. Либо нажмите клавиши **CTRL**+**ALT**+**X**.)

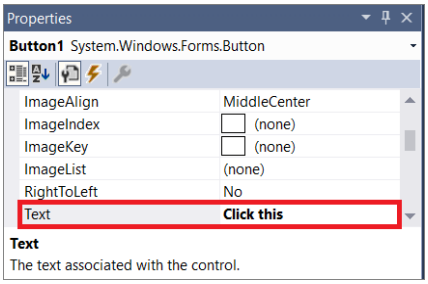
2. Щелкните значок **Закрепить**, чтобы закрепить окно **Панель элементов**.



3. Выберите элемент управления **Кнопка** и перетащите его на форму.

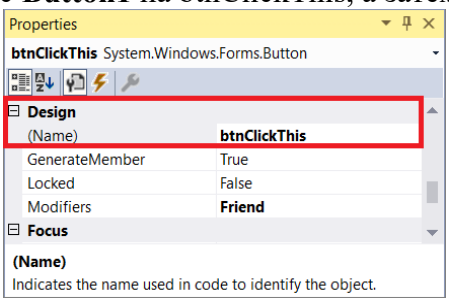


4. В окне **Свойства** найдите элемент **Текст**, измените имя с **Button1** на Click this, а затем нажмите клавишу **ВВОД**.



(Если окно **Свойства** не отображается, его можно открыть в строке меню.) Для этого выберите **Вид** > **Окно свойств**. Или нажмите клавишу **F4**.)

5. В разделе **Проектирование** окна **Свойства** измените имя с **Button1** на btnClickThis, а затем нажмите клавишу **ВВОД**.



Примечание

Если список был упорядочен по алфавиту в окне **Свойства**, **Button1** появится в разделе **(DataBindings)** .

Добавление метки на форму

Теперь, когда мы добавили элемент управления "Кнопка" для создания действия, давайте добавим элемент управления "Метка", куда можно отправлять текст.

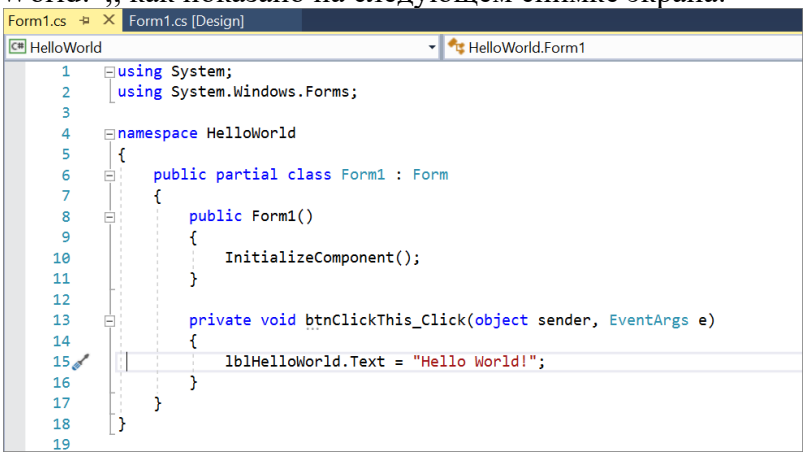
1. Выберите элемент управления **Метка** в окне **Панель элементов**, а затем перетащите его на форму и расположите под кнопкой **Нажмите это**.
2. В разделе **Проект** или **(DataBindings)** окна **Свойства** измените имя **Label1** на lblHelloWorld и нажмите клавишу **ВВОД**.

Добавление кода на форму

1. В окне **Form1.cs [Проект]** дважды щелкните кнопку **Нажмите это**, чтобы открыть окно **Form1.cs**.

(Кроме того, можно развернуть узел **Form1.cs** в **обозревателе решений**, а затем выбрать **Form1**.)

2. В окне **Form1.cs** после строки **private void** введите lblHelloWorld.Text = "Hello World!";, как показано на следующем снимке экрана:

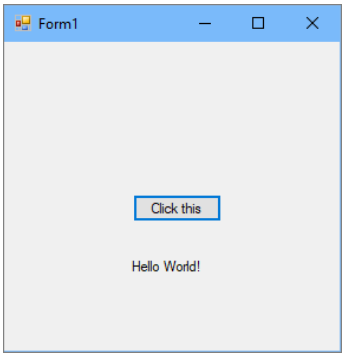


Запуск приложения

1. Нажмите кнопку **Запустить**, чтобы запустить приложение.

Будет выполнено несколько операций. В интегрированной среде разработки Visual Studio откроются окна **Средства диагностики** и **Вывод**. Кроме того, вне этой среды откроется диалоговое окно **Form1**. Оно будет содержать вашу кнопку **Нажмите это** и текст **Label1**.

2. Нажмите кнопку **Нажмите это** в диалоговом окне **Form1**. Обратите внимание, что текст **Label1** меняется на **Hello World!** .



3. Закройте диалоговое окно **Form1**, чтобы завершить работу приложения.

Практическая работа №22 Разработка игрового приложения

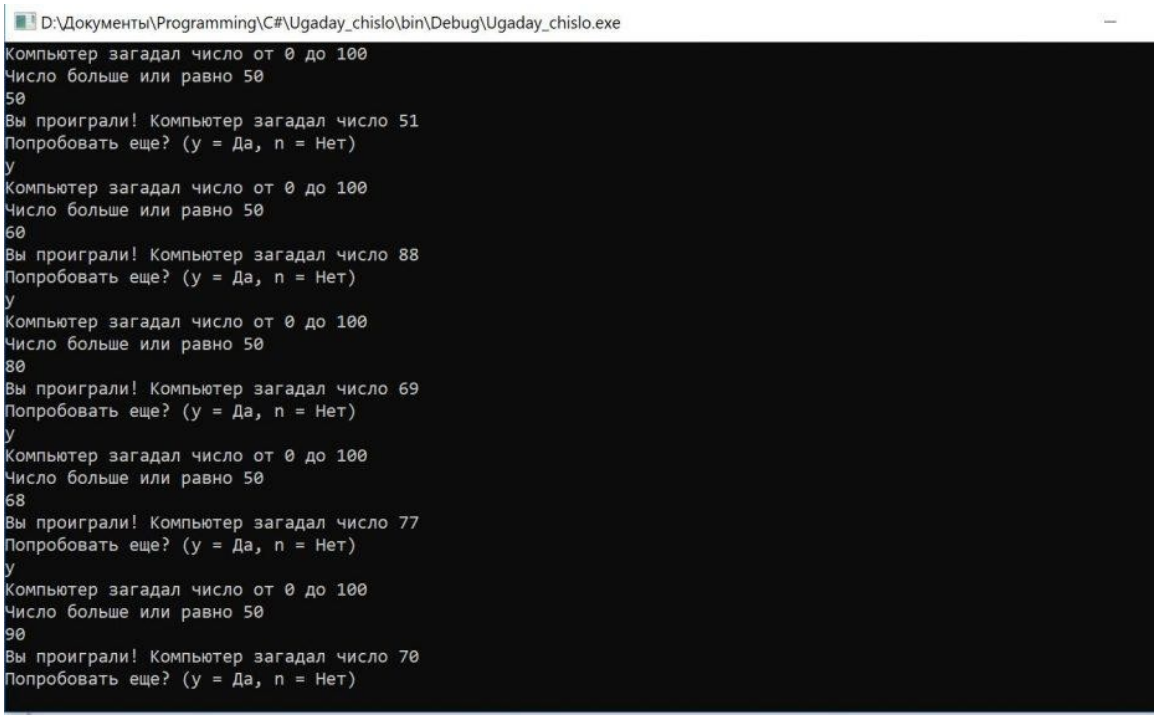
разработаем простую мини игру “Угадай число” на языке C#. Суть игры простой: компьютер загадывает число от 0 до 100, выдаст подсказку – больше ли это число 50 или нет, затем сравнит введенное пользователем число с загаданным. Загадывание числа будет реализовано с помощью генератора случайных чисел:

```
Random rand = new Random();
int i = rand.Next(100);
```

Число “100” можно поменять. Если задать число “50”, тогда компьютер будет загадывать от 0 до 50. Чтобы создать игру зайдите в Visual Studio, создайте проект “Консольное приложение (.NET Framework)” на языке C# и перепишите код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Ugaday_chislo
    {
      class Program
      {
        static void Main(string[] args)
        {
          Console.OutputEncoding = Encoding.GetEncoding(866);
          Console.InputEncoding = Encoding.GetEncoding(866);

          char again = 'y';
          Random rand = new Random();

          while (again == 'y')
          {
            int i = rand.Next(100);

            Console.WriteLine("Компьютер загадал число от 0 до 100");

            if (i < 50) Console.WriteLine("Число меньше 50");
            else Console.WriteLine("Число больше или равно 50");

            int x = Convert.ToInt32(Console.ReadLine());

            if (i == x) Console.WriteLine("Поздравляем! Вы победили свой компьютер!");
            else Console.WriteLine("Вы проиграли! Компьютер загадал число {0}", i);

            Console.WriteLine("Попробовать еще? (y = Да, n = Нет)");
            again = Convert.ToChar(Console.ReadLine());

          }
        }
      }
    }
```

Результат программы:



Внимательно изучите код и попробуйте усовершенствовать его.

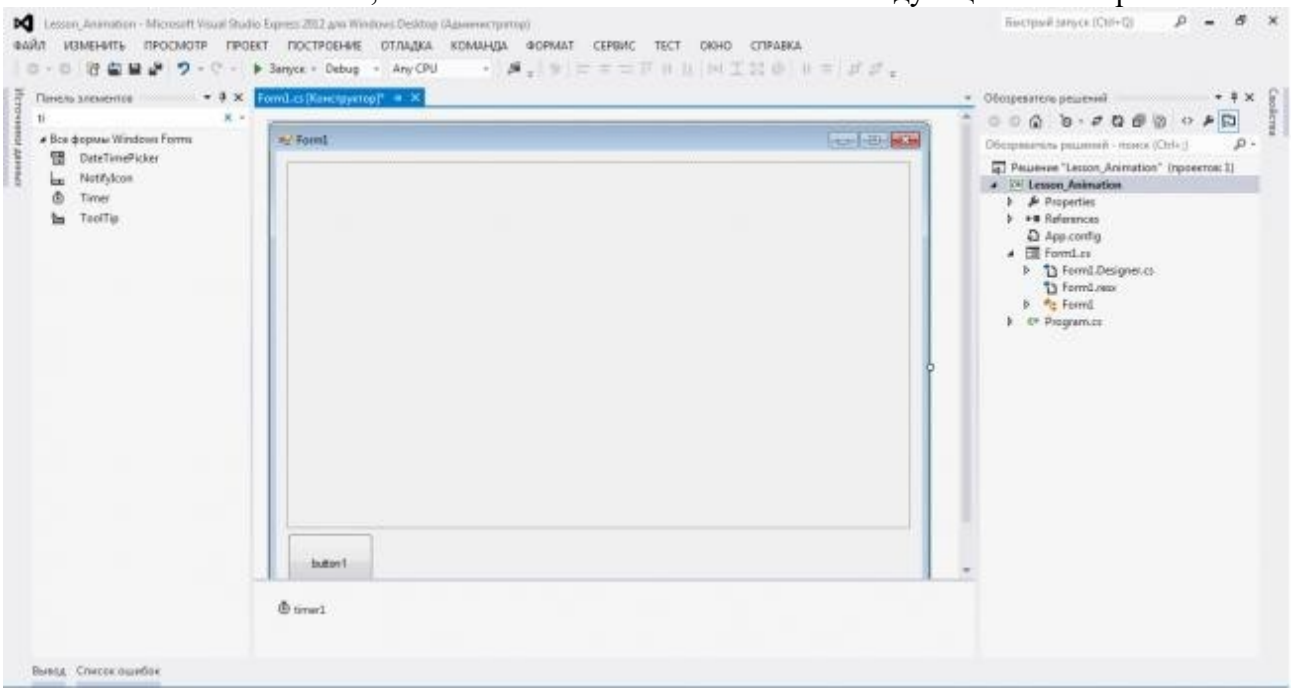
Практическая работа №23 Разработка приложения с анимацией

оздание анимации в C#

В данном уроке используется среда программирования Microsoft Visual studio 2012. Алгоритм работы аналогичен и для других сред программирования. Перед тем, как приступить к данному уроку, следует ознакомиться с предыдущим уроком

: Как начать работать с графикой в Microsoft C#

Сначала создаем свой проект, в котором и будем работать. Разместим на форме (Form1) объекты : PictureBox, Timer и Button следующим образом:



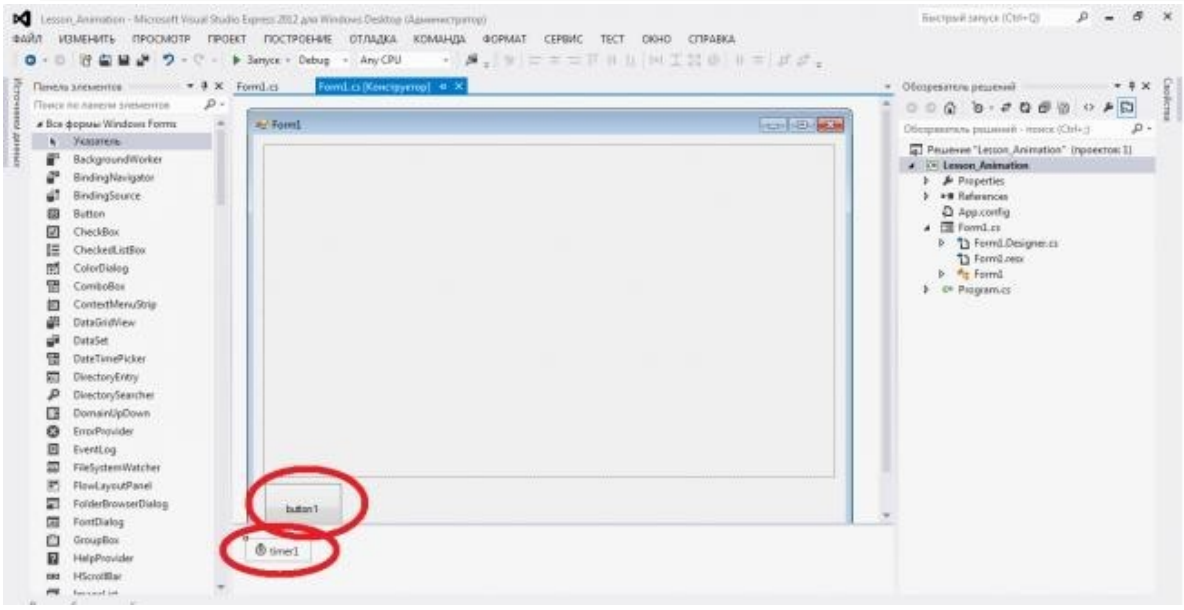
Теперь попробуем создать какую-нибудь примитивную анимацию, при помощи PictureBox и Timer, которая запустится после нажатия на кнопку (Button). Следовательно для этого будем обрабатывать событие нажатия на кнопку и событие "срабатывания" таймера. Также заведем все нужные для рисования объекты и переменные.

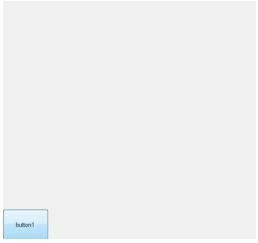
Далее приведен код программы и скриншоты, которые содержат все необходимое пояснение реализации анимации.

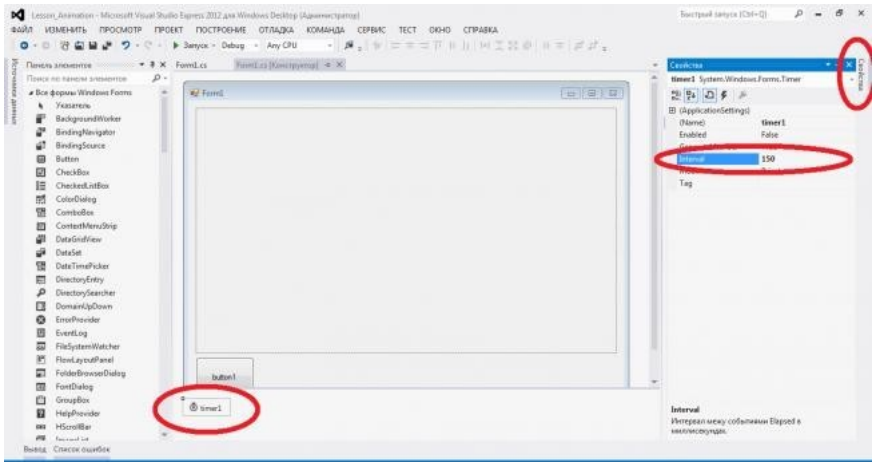
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Lesson_Animation
{
   public partial class Form1 : Form
   {
      Graphics gr;      //объявляем объект - графику, на которой будем рисовать
      Pen p;            //объявляем объект - карандаш, которым будем рисовать контур
      SolidBrush fon;   //объявляем объект - заливки, для заливки соответственно фона
      SolidBrush fig;   //и внутренности рисуемой фигуры
```

```
int rad;         // переменная для хранения радиуса рисуемых кругов
Random rand;     // объект, для получения случайных чисел

public Form1()
{
   InitializeComponent();
}

//опишем функцию, которая будет рисовать круг по координатам его центра
void DrawCircle(int x, int y)
{
   int   xc,   yc;
   xc = x - rad;
   yc = y - rad;
   gr.FillEllipse(fig, xc, yc, rad, rad);

   gr.DrawEllipse(p, xc, yc, rad, rad);

}

// для перехода к данной функции сделайте двойной щелчок по кнопке (Button)
// добавленной на форму. См. на фото, после кода
private void button1_Click(object sender, EventArgs e)
{
   gr = pictureBox1.CreateGraphics();  //инициализируем объект типа графики
                                       // привязав  к PictureBox

   p = new Pen(Color.Lime);          // задали цвет для карандаша
   fon = new SolidBrush(Color.Black); // и для заливки
   fig = new SolidBrush(Color.Purple);

   rad = 40;                        //задали радиус для круга
   rand = new Random();             //инициализируем объект для рандомных числе

   gr.FillRectangle(fon, 0, 0, pictureBox1.Width, pictureBox1.Height); // закрасим черным
                                                // нашу область рисования

   // вызываем написанную нами функцию, для прорисовки круга
   // случайным образом выбрав перед этим координаты центра
   int x,  y;

   for (int i = 0; i < 15; i++)
   {
      x = rand.Next(pictureBox1.Width);
      y = rand.Next(pictureBox1.Height);
      DrawCircle(x, y);
   }

   timer1.Enabled = true;  //включим в работу наш таймер,
   // то есть теперь будет происходить событие Tick и его будет обрабатывать
функция On_Tick (по умолчанию)
```

```
        }

        // для получения данной функции перейдите к конструктору формы
        // и сделайте двойной щелчок по таймеру, добавленному на форму. См. на фото после кода
        private void timer1_Tick(object sender, EventArgs e)
        {

            //сначала будем очищать область рисования цветом фона
            gr.FillRectangle(fon, 0, 0, pictureBox1.Width, pictureBox1.Height);


            // затем опять случайным образом выбираем координаты центров кругов
            // и рисуем их при помощи описанной нами функции
            int x, y;

            for (int i = 0; i < 15; i++)
            {
                x = rand.Next(pictureBox1.Width);
                y = rand.Next(pictureBox1.Height);
                DrawCircle(x, y);
            }

        }
    }
}
```







После этого, можно запустить программу и после нажатия на кнопку увидите простую анимацию – случайное перемещение кругов по черному фону, как показано ниже:
Для того, чтобы анимация соответствовала требованиям иногда необходимо менять так называемый тик таймера, т.е. промежуток выполнения очередного шага анимации. Это выполняется в Инспекторе объектов. Нужно выбрать элемент Timer, нажать на кнопку Свойства и там выбрать и изменить параметр Interval (выражается в миллисекундах) В данном примере Interval равен 150 мс.

Практическая работа №24 Оптимизация и рефакторинг кода

Рефакторинг- своего рода перепроектирования кода программы, уменьшив ее в объеме но не изменив ее функциональности, порой после рефакторинга код программы может сократиться в десятки раз. И так рассмотрим код метода.

```
1  static bool ShouldFire(bool enemyInFront, string enemyName, int robotHealth)
2  {
3          bool shouldFire = true;
4          if (enemyInFront == true)
5          {
6                  if (enemyName == "boss")
7                  {
8                          if (robotHealth < 50) shouldFire = false;
9                          if (robotHealth > 100) shouldFire = true;
10                 }
11         }
12         else
13         {
14                 return false;
15         }
16         return shouldFire;
17 }
```

Посмотрев на код выше, сразу бросается количество ветвлений условных операторов. И относительно не большая функциональность метода, занимает слишком много строк кода. Попробуйте самостоятельно уменьшить ее размер и если ли не получиться подстмотрите код ниже. Функциональность метода нельзя нарушать, лишь сократить ее в объеме. В первую очередь для меня бросается локальная переменная shouldFire которая тут вообще не нужна. А весь остальной код можно сгруппировать в return. И так смотрите что у меня получилось:

```
1  static bool ShouldFire2(bool enemyInFront, string enemyName, int robotHealth)
2                  {
3                          return enemyInFront? ((enemyName=="boss")&&
4  robotHealth<50?false:true):false;
                   }
```

Метод настолько уменьшился что поместился в одну строку, а функциональность его осталось не изменой. Рефакторинг программы следует проводить только тогда, кода ваш код рабочий, не имеет ошибок. Только после этого надо думать над тем как его уменьшить. Но сильно уменьшать его тоже не стоит, так как сокращение кода может вести к сложности читаемости кода, в особенности если над кодом работать будут и другие разработчики.

При рефакторинге стоит учитывать то, что метод должен вмещаться в 20 строк кода и максимум в один экран, в случаи разрастания кода следует метод разделять на под методы, это уменьшит их, упростит процедуру отладки, а другим программистам сократит время на понимание логики. Так же при работе с операторами выбора, такими как switch

не стоит злоупотреблять, в программе, а лучше всего их заменить на словари Directionary, а так же Делегаты.

Используйте StringBuilder над String, чтобы получить лучшую производительность:

Есть целый ряд статей и сообщений , которые говорят , что StringBuilderявляется более эффективным , поскольку он содержит изменяемый буфер строки. .NET Строки неизменны, что является причиной , почему новый Stringобъект создается каждый раз , когда мы изменяем его (вставка, Append, удалить и т.д.).

В следующем разделе я объясню это более подробно, чтобы дать новичкам четкое представление об этом факте.

Я написал следующий код, и , как вы можете видеть, я определил Stringпеременную и StringBuilderпеременную. Я затем добавить строку в обоих этих переменных:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace StringTest
7  {
8     class Program
9     {
10       static void Main(string[] args)
11       {
12          string s = "Pranay";
13          s += " rana";
14          s += " rana1";
15          s += " rana122";
16
17          StringBuilder sb = new
18          StringBuilder(); sb.Append("pranay");
19          sb.Append(" rana");
20       }
21    }
22 }
```

Если разобрать этот код под рефлетором станет понятно по какой причине лучше использовать StringBuilder.

Если вы зашли в рефлектор можете видеть, Concatфункция принимает два аргумента и возвращает String. Следующие шаги выполняются , когда мы выполняем Append с типом строки:

1. Проверяет, является ли строка пустой или нет
2. Создает строку destи выделяет память для строки
3. Заполняет destстроку str0иstr1
4. Возвращает destстроку, которая представляет собой новую переменную строку

Таким образом, это доказывает, что всякий раз, когда я делаю операцию конкатенации строк, он создает новую строку из-за неизменное поведение строк.

Функция Append принимает аргумент типа Stringи возвращает StringBuilderобъект. Следующие шаги выполняются , когда мы выполняем Appendс StringBuilderтипом:

1. Получить значение строки из StringBuilderобъекта
2. Проверьте, если требуется выделить память для новой строки мы будем добавлять

3. Выделение памяти при необходимости и добавить строку
4. Если не требуется, чтобы выделить память, а затем добавить строку непосредственно к существующей выделенной памяти
5. Возвращает StringBuilderобъект , который вызывает функцию, используя thisключевое слово

Таким образом, она возвращает тот же объект, не создавая новый.

Я надеюсь , что этот пример помог вам понять внутренние детали о том, почему мы должны использовать StringBuilderчаще чем , Stringчтобы получить более высокую производительность , когда мы делаем основную обработку строк в коде.

Структура инициализации в C #

Структуры в C # позволяют нам группировать переменные и методы. Они несколько похожи на классы, но есть ряд отличий между ними. В этой статье я не буду обсуждать это. Я собираюсь объяснить, как инициализировать структуру.

8. **факты**

1. Структура является типом значения
2. Это не позволяет создать параметр меньше конструктора, поскольку он инициализирует переменную со значениями по умолчанию

Теперь рассмотрим ниже случай, когда я создал две структуры:

9. **Структура 1: с открытыми переменными.**

```
1  public struct StructMember
2  {
3     public int  a;
4     public int  b;
5  }
```

10. **Структура 2: со свойствами.**

```
1   public struct StructProperties
2   {
3         private int a;
4         private int b;
5
6         public int A
7         {
8               get
9               { return a; }
10              set
11              { a = value; }
12        }
13
14        public int B
15        {
16              get
17              { return b; }
18              set
```

```
19              { b = value; }
20      }
21  }
```

Учитывая вышеуказанные два факта, я пытался использовать обе структуры, как показано ниже:

```
1   public class MainClass
2   {
3      public static void Main()
4      {
5        StructMembers MembersStruct;
6
7        StructProperties PropertiesStruct;
8
9        MembersStruct.X = 100;
10       MembersStruct.Y = 200;
11
12       PropertiesStruct.X = 100;
13       PropertiesStruct.Y = 200;
14     }
15  }
```

После этого, когда я пытаюсь скомпилировать код, я получаю сообщение об ошибке.
C # компилятор сообщает нам, что позволяет использовать первую структуру без ошибок, но не позволяет использовать вторую структуру, которая выставляет свойство. Чтобы решить эту проблему, я написал ниже строки кода для инициализации второй структуры:

```
1 StructProperties PropertiesStruct = new  StructProperties();
```

Дело в том, что структура может быть реализована без использования newоператора. Если вы не используете new, то поля будут оставаться Unassigned и объект не может использоваться , пока все поля не инициализированы.
В .NET, все простые типы структур когда вы пишете код на C # необходимо инициализировать, рассмотрим случай ниже, где я создаю целочисленную переменную:

```
1  int a;
2  Console.WriteLine(a);
```

Компилятор выдает ошибку, что вы не можете использовать переменную без инициализации. Так что вам нужно написать либо:

```
1  int a =0;
```

или же
Используйте новый оператор для вызова конструктора по умолчанию и присвоить значение по умолчанию для переменной

```
<span class="code-keyword">int</span> a = <span class="code-keyword">new</span> <span
class="code-keyword">int</span>();
```

Это очень важно, чтобы инициализировать структуру правильно.

Оператор Checked

В следующем разделе я буду объяснять о Checked оператора доступна в C # .NET для обработки целочисленных переполнений.
В моей системе управления заказами, я должен вычислить точку продаж для каждого клиента, который размещает заказ. Пункты продажи являются целыми числами, которые получают заработанные клиент на основе продуктов, которые они покупают, и получает вычитаются из общей суммы счета, как они покупают новые продукты, используя эти точки. Но есть некоторые клиенты, которые не знают о балльной системе или не потребляют эти точки так, что точки накапливают более чем предел целочисленных переменных, то есть, 2 ^

32. Таким образом, всякий раз, когда расчет происходит, я получаю некоторое значение опасное для тех клиентов, которые имеют значение точек больше, чем максимально допустимые целочисленное значение.
Чтобы избежать этой проблемы переполнения целых значений и информировать клиентов о своих точках, я использую Checked оператор C # .NET.
Проверено оператор для проверки на переполнение в математических операциях и переходах для целочисленных типов.

11. Синтаксис

```
1  Checked( expression )
1  Checked { statements.....}
```

```
1   public static void Main()
2   {
3      int a;
4      int b;
5      int c;
6
7      a = 2000000000;
8      b = 2000000000;
9      c = checked(a+ b); System.Console.Write-
10     Line(Int1PlusInt2);
11  }
```

Когда мы запустим код выше, он бросает исключение.
который говорит , что сбольше , чем максимально допустимое целочисленное значение. Точно так же, в моем приложении, я ловлю исключение переполнения брошенного при расчете точке заказа, с и затем отправить почту клиенту с напоминанием о необходимости использовать очки.

GO TO Switch..Case

12. Go To Идти к..

Позволяет нам прыгать безоговорочно, когда это требуется и не рекомендуется для использования в большой степени.

13. Switch..Case

Позволяет выполнить Caseблок на основе значения переменной, то есть, позволяет сделать программирование на основе условий.
В моем приложении, я достиг той стадии , когда я должен был выполнить код Case 1из Switch..Caseи если какое — то условие было выполнено, я должен был выполнить код Case 3из Switch..Case.

```
      Switch(myvariable)
2   {
3      case 1:
4         //statements
5         ….....
6         if ( expression )
7            execute case 3:
8         break;
9
10     case 2:
11        …..
12        break;
13
14     case 3:
15        ….....
```

```
16        break;
17  }
```

Первое решение этой проблемы , чтобы скопировать код Case 3и поместить его в ifблоке Case 1.

```
1  case 1:
2     //statements
3     ….....
4     if ( expression )
5       //code of case 3
6     break;
```

Но проблема выше решения заключается в том, что он делает избыточный код.

Второе решение заключается в создании функции и поместить код в том , а затем выполнить код.

```
1    case 1:
2       //statements
3       ….....
4       if ( expression )
5       Case3Code();
6       break;
7
8    function Case3Code()
9    {
10    ….
11   }
```

Проблема с этим решением является то, что я должен создать дополнительную функцию, которая не нужна.

Третье решение , чтобы сделать использование Go Тов Switch..Caseблоке:

```
1    switch(MyVariable)
2    {
3       case 1:
4          //statements
5          ….....
6          if ( expression )
7             goto case 3:
8          break;
9
10      case 2:
11         …..
12         break;
13
14      case 3:
15         ….....
16         break;
17   }
```

Go toB Switch..Caseпозволяет мне сделать код легко и в обслуживаемой образом.

Практическая работа №26 Создание приложения с БД

Создание простого приложения для работы с данными с помощью ADO.NET

При создании приложения, которое работает с данными в базе данных, необходимо выполнить такие основные задачи, как определение строк подключения, вставка данных и выполнение хранимых процедур. В этом разделе вы узнаете, как взаимодействовать с базой данных из простого Windows Forms приложения "формы по данным" с помощью Visual C#, Visual Basic и ADO.NET. Все технологии данных .NET, в том числе наборы данных, LINQ to SQL и Entity Framework, в конечном итоге выполняют шаги, которые очень похожи на те, которые приведены в этой статье.

В этой статье демонстрируется простой способ быстрого получения данных из базы данных. Если приложению необходимо изменить данные с помощью нетривиальных способов и обновить базу данных, следует рассмотреть возможность использования Entity Framework и привязки данных для автоматической синхронизации элементов управления пользовательского интерфейса с изменениями в базовых данных.

Важно!

С целью упрощения код не включает обработку исключений для выполнения в рабочей среде.

Предварительные требования

Для создания приложения вам потребуются следующие компоненты.

- приведенному.
- SQL Server Express LocalDB. Если у вас нет SQL Server Express LocalDB, его можно установить на странице загрузки SQL Server Express.

В этом разделе предполагается, что вы знакомы с базовой функциональностью интегрированной среды разработки Visual Studio и можете создать Windows Forms приложение, добавить формы в проект, поместить кнопки и другие элементы управления в формы, задать свойства элементов управления и создать код для простых событий. Если вы не знакомы с этими задачами, мы рекомендуем выполнить инструкции по началу работы с Visual C# и Visual Basic , прежде чем приступать к этому пошаговому руководству.

Настройка образца базы данных

Создайте образец базы данных, выполнив следующие действия.

1. В Visual Studio откройте окно **Обозреватель сервера** .
2. Щелкните правой кнопкой мыши **подключения к данным** и выберите команду **создать новую базу данных SQL Server**.
3. В текстовом поле **имя сервера** введите **(LocalDB) \mssqllocaldb**.
4. В текстовом поле **имя новой базы данных** введите **Sales**, а затем нажмите кнопку **ОК**.

Пустая база данных **Sales** создается и добавляется в узел подключения к данным в обозреватель сервера.

5. Щелкните правой кнопкой мыши подключение к данным о **продажах** и выберите **создать запрос**.

Откроется окно редактора запросов.

6. Скопируйте скрипт Transact-SQL Sales в буфер обмена.
7. Вставьте скрипт T-SQL в редактор запросов, а затем нажмите кнопку **выполнить** .

По истечении короткого времени выполнение запроса завершается и создаются объекты базы данных. База данных содержит две таблицы: Customer и Orders. Эти таблицы изначально не содержат данных, но их можно добавить при запуске создаваемого приложения. База данных также содержит четыре простые хранимые процедуры.

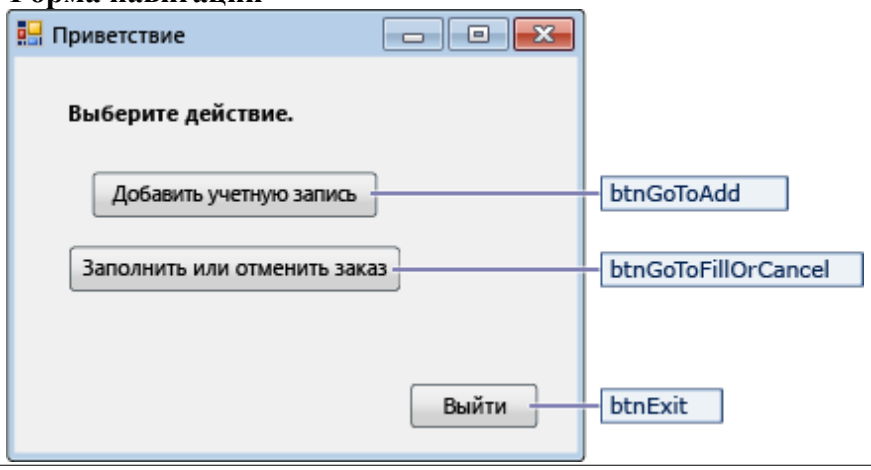
Создание форм и добавление элементов управления

1. Создайте проект для приложения Windows Forms и назовите его **SimpleDataApp**. Visual Studio создает проект и несколько файлов, включая пустую форму Windows Forms с именем **Form1**.
2. Добавьте две формы Windows Forms в проект, чтобы он включал три формы, и назначьте им следующие имена:
   - **Навигация**
   - **NewCustomer**
   - **FillOrCancel**
3. Для каждой формы добавьте текстовые поля, кнопки и другие элементы управления, которые отображаются на рисунках ниже. Для каждого элемента управления задайте свойства, указанные в таблицах.

Примечание

Элементы управления "группа" и "надпись" обеспечивают большую ясность, но не используются в коде.

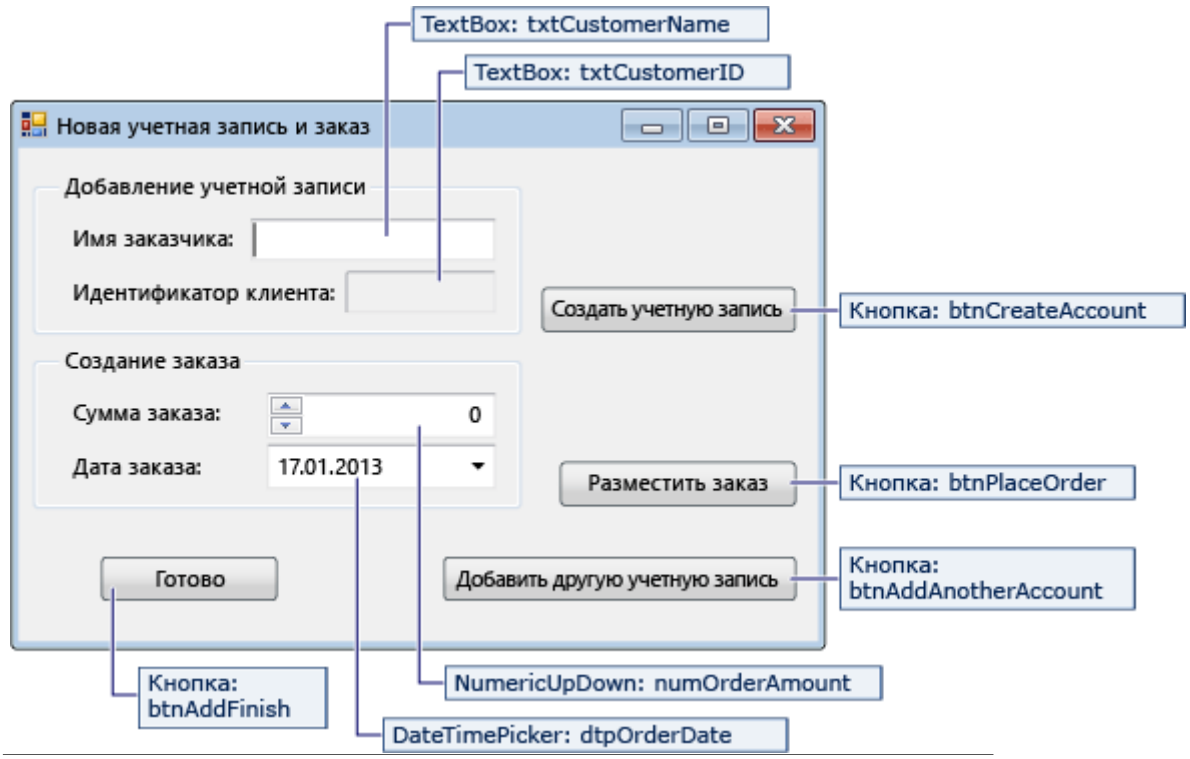
Форма навигации



СОЗДАНИЕ ФОРМ И ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Элементы управления формы навигации	Свойс
Кнопка	Name
Кнопка	Name
Кнопка	Name

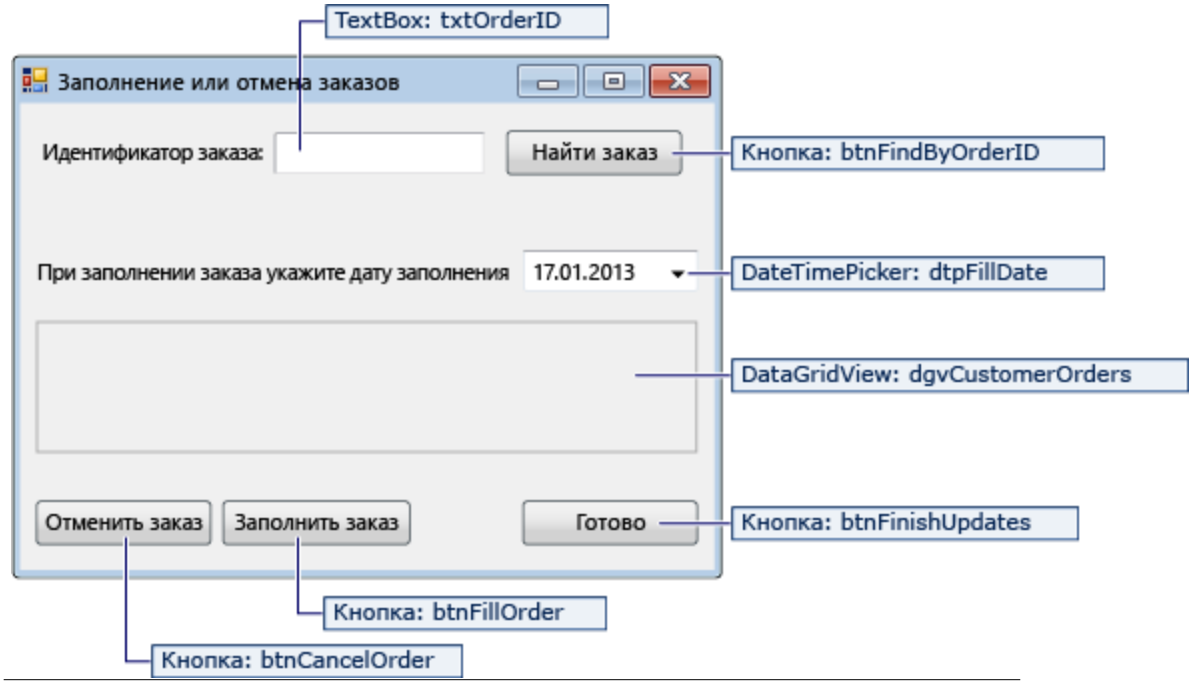
Форма NewCustomer



СОЗДАНИЕФОРМ И ДОБАВЛЕНИЕ ЭЛЕМЕНТОВУПРАВЛЕНИЯ

Элементы управления формы NewCustomer	**Свойства**
TextBox	Name = txtCustomerName
TextBox	Name=txtCustomerID Readonly = True
Кнопка	Name = btnCreateAccount
NumericUpDown	DecimalPlaces=0 Maximum=5000 Name = numOrderAmount
DateTimePicker	Format=Short Name = dtpOrderDate
Кнопка	Name = btnPlaceOrder
Кнопка	Name = btnAddAnotherAccount
Кнопка	Name = btnAddFinish

Форма FillOrCancel



Т А Б Л И Ц А 3

Элементы управления формы FillOrCancel	**Свойства**
TextBox	Name = txtOrderID
Кнопка	Name = btnFindByOrderID
DateTimePicker	Format=Short Name = dtpFillDate
DataGridView	Name=dgvCustomerOrders Readonly=True RowHeaders Visible = False
Кнопка	Name = btnCancelOrder
Кнопка	Name = btnFillOrder
Кнопка	Name = btnFinishUpdates

Сохранение строки подключения

Когда приложение пытается открыть подключение к базе данных, оно должно иметь доступ к строке подключения. Чтобы не вводить строку вручную в каждой форме, сохраните строку в файле *App.config* в проекте и создайте метод, возвращающий строку при вызове метода из любой формы в приложении.

Строку подключения можно найти, щелкнув правой кнопкой мыши подключение данных о **продажах** в **Обозреватель сервера** и выбрав **Свойства**. Найдите свойство **ConnectionString** , а затем с помощью **клавиш CTRL + A**, **CTRL + C** выберите и скопируйте строку в буфер обмена.

1. Если вы используете C#, в **Обозреватель решений** разверните узел **свойства** в проекте, а затем откройте файл **Settings. Settings** . Если вы используете Visual Basic, в **Обозреватель решений** выберите пункт **Показывать все файлы**, разверните узел **Мой проект** , а затем откройте файл **Settings. Settings** .
2. В столбце **имя** введите connString .
3. В списке **тип** выберите **(строка подключения)**.

4. В списке **область** выберите **приложение**.
5. В столбце **значение** введите строку подключения (без кавычек), а затем сохраните изменения.

Примечание

В реальных приложениях строку подключения следует хранить безопасно, как описано в разделе **строки подключения и файлы конфигурации**.

Написание кода для форм

Этот раздел содержит краткие обзоры того, что делает каждая форма. Он также предоставляет код, определяющий базовую логику при нажатии кнопки на форме.

Форма навигации

Форма навигации открывается при запуске приложения. Кнопка **Добавить учетную запись** открывает форму NewCustomer. Кнопка **Выполнение или отмена заказов** открывает форму FillOrCancel. Кнопка **Выход** закрывает приложение.

14. Преобразование формы навигации в начальную форму

При использовании C# в **обозревателе решений** откройте файл **Program.cs** и измените строку Application.Run на следующую: Application.Run(new Navigation());

Если вы используете Visual Basic, в **Обозреватель решений** откройте окно **свойства** , перейдите на вкладку **приложение** и выберите **симпледатаапп. Navigation** в списке **начальных форм** .

15. Создание автоматически создаваемых обработчиков событий

Дважды щелкните три кнопки в форме навигации, чтобы создать пустые методы обработчика событий. При двойном щелчке кнопки также добавляется автоматически созданный код в файл кода конструктора, который позволяет нажать кнопку для вызова события.

16. Добавление кода для логики формы навигации

На странице кода для формы навигации заполните основные тексты методов для трех обработчиков событий нажатия кнопки, как показано в следующем коде.

C#Копировать

```
/// <summary>
/// Opens the NewCustomer form as a dialog box,
/// which returns focus to the calling form when it is closed.
/// </summary>
private void btnGoToAdd_Click(object sender, EventArgs e)
    {
      Form frm = new NewCustomer();
      frm.Show();
    }

/// <summary>
/// Opens the FillorCancel form as a dialog box.
/// </summary>
private void btnGoToFillOrCancel_Click(object sender, EventArgs e)
    {
      Form frm = new FillOrCancel();
      frm.ShowDialog();
    }

/// <summary>
/// Closes the application (not just the Navigation form).
/// </summary>
private void btnExit_Click(object sender, EventArgs e)
    {
```

```
    this.Close();
  }
```

Форма NewCustomer

Если ввести имя клиента, а затем нажать кнопку **создать учетную запись** , форма newCustomer создает учетную запись клиента, а SQL Server ВОЗВРАЩАЕТ значение идентификатора в качестве нового идентификатора клиента. Затем можно разместить заказ для новой учетной записи, указав сумму и дату заказа и нажав кнопку **поместить порядок** .

17. Создание автоматически создаваемых обработчиков событий

Создайте пустой обработчик событий щелчка для каждой кнопки в форме NewCustomer, дважды щелкнув каждую из четырех кнопок. При двойном щелчке кнопки также добавляется автоматически созданный код в файл кода конструктора, который позволяет нажать кнопку для вызова события.

18. Добавление кода для логики формы NewCustomer

Чтобы завершить логику формы NewCustomer, выполните следующие действия.

1. Перенесите System.Data.SqlClient пространство имен в область, чтобы не указывать полные имена его членов.

C#Копировать

```
using System.Data.SqlClient;
```

2. Добавьте в класс некоторые переменные и вспомогательные методы, как показано в следующем коде.

C#Копировать

```
// Storage for IDENTITY values returned from database.
private int parsedCustomerID;
private int orderID;

/// <summary>
/// Verifies that the customer name text box is not empty.
/// </summary>
private bool IsCustomerNameValid()
  {
    if (txtCustomerName.Text == "")
    {
      MessageBox.Show("Please enter a name.");
      return false;
    }
    else
    {
      return true;
    }
  }

/// <summary>
/// Verifies that a customer ID and order amount have been provided.
/// </summary>
private bool IsOrderDataValid()
  {
    // Verify that CustomerID is present.
    if (txtCustomerID.Text == "")
    {
      MessageBox.Show("Please create customer account before placing order.");
      return false;
```

```
        }
        // Verify that Amount isn't 0.
        else if ((numOrderAmount.Value < 1))
        {
            MessageBox.Show("Please specify an order amount.");
            return false;
        }
        else
        {
            // Order can be submitted.
            return true;
        }
    }

/// <summary>
/// Clears the form data.
/// </summary>
private void ClearForm()
    {
        txtCustomerName.Clear();
        txtCustomerID.Clear();
        dtpOrderDate.Value = DateTime.Now;
        numOrderAmount.Value = 0;
        this.parsedCustomerID = 0;
    }
```

3. Заполните основные тексты методов для четырех обработчиков событий нажатия кнопки, как показано в следующем коде.

C#Копировать

```
/// <summary>
/// Creates a new customer by calling the Sales.uspNewCustomer stored procedure.
/// </summary>
private void btnCreateAccount_Click(object sender, EventArgs e)
    {
        if (IsCustomerNameValid())
        {
            // Create the connection.
       using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
            {
                // Create a SqlCommand, and identify it as a stored procedure.
           using (SqlCommand sqlCommand = new SqlCommand("Sales.uspNewCustomer", connection))
                {
                    sqlCommand.CommandType = CommandType.StoredProcedure;

                    // Add input parameter for the stored procedure and specify what to use as its value.
                    sqlCommand.Parameters.Add(new SqlParameter("@CustomerName", SqlDbType.NVarChar, 40));
                    sqlCommand.Parameters["@CustomerName"].Value = txtCustomerName.Text;

                    // Add the output parameter.
                    sqlCommand.Parameters.Add(new SqlParameter("@CustomerID", SqlDbType.Int));
```

```
                sqlCommand.Parameters["@CustomerID"].Direction = ParameterDirection.Output;

                try
                {
                    connection.Open();

                    // Run the stored procedure.
                    sqlCommand.ExecuteNonQuery();

                    // Customer ID is an IDENTITY value from the database.
                    this.parsedCustomerID = (int)sqlCommand.Parameters["@CustomerID"].Value;

                    // Put the Customer ID value into the read-only text box.
                    this.txtCustomerID.Text = Convert.ToString(parsedCustomerID);
                }
                catch
                {
                    MessageBox.Show("Customer ID was not returned. Account could not be
    created.");
                }
                finally
                {
                    connection.Close();
                }
            }
        }
    }
}

/// <summary>
/// Calls the Sales.uspPlaceNewOrder stored procedure to place an order.
/// </summary>
private void btnPlaceOrder_Click(object sender, EventArgs e)
    {
        // Ensure the required input is present.
        if (IsOrderDataValid())
        {
            // Create the connection.
       using (SqlConnection connection = new
SqlConnection(Properties.Settings.Default.connString))
            {
                // Create SqlCommand and identify it as a stored procedure.
           using (SqlCommand sqlCommand = new SqlCommand("Sales.uspPlaceNewOrder", connection))
                {
                    sqlCommand.CommandType = CommandType.StoredProcedure;

                    // Add the @CustomerID input parameter, which was obtained from uspNewCustomer.
                    sqlCommand.Parameters.Add(new SqlParameter("@CustomerID", SqlDbType.Int));
                    sqlCommand.Parameters["@CustomerID"].Value = this.parsedCustomerID;

                    // Add the @OrderDate input parameter.
```

```
            sqlCommand.Parameters.Add(new SqlParameter("@OrderDate",
SqlDbType.DateTime, 8));
                sqlCommand.Parameters["@OrderDate"].Value = dtpOrderDate.Value;

                // Add the @Amount order amount input parameter.
                sqlCommand.Parameters.Add(new SqlParameter("@Amount", SqlDbType.Int));
                sqlCommand.Parameters["@Amount"].Value = numOrderAmount.Value;

                // Add the @Status order status input parameter.
                // For a new order, the status is always O (open).
                sqlCommand.Parameters.Add(new SqlParameter("@Status", SqlDbType.Char, 1));
                sqlCommand.Parameters["@Status"].Value = "O";

                // Add the return value for the stored procedure, which is the order ID.
                sqlCommand.Parameters.Add(new SqlParameter("@RC", SqlDbType.Int));
                sqlCommand.Parameters["@RC"].Direction = ParameterDirection.ReturnValue;

                try
                {
                  //Open connection.
                  connection.Open();

                  // Run the stored procedure.
                  sqlCommand.ExecuteNonQuery();

                  // Display the order number.
                  this.orderID = (int)sqlCommand.Parameters["@RC"].Value;
                  MessageBox.Show("Order number " + this.orderID + " has been submitted.");
                }
                catch
                {
                  MessageBox.Show("Order could not be placed.");
                }
                finally
                {
                  connection.Close();
                }
              }
            }
          }
        }

/// <summary>
/// Clears the form data so another new account can be created.
/// </summary>
private void btnAddAnotherAccount_Click(object sender, EventArgs e)
    {
      this.ClearForm();
    }

/// <summary>
/// Closes the form/dialog box.
```

```
/// </summary>
private void btnAddFinish_Click(object sender, EventArgs e)
    {
      this.Close();
    }
```

Форма FillOrCancel

Форма Филлорканцел запускает запрос для возврата заказа при вводе идентификатора заказа и нажатия кнопки **найти заказ** . Возвращенная строка отображается в сетке данных только для чтения. Можно пометить заказ как отмененный (X), если нажать кнопку **отменить заказ** или пометить заказ как заполненный (F), если нажать кнопку **заполнить заказ** . Если нажать кнопку **найти порядок** еще раз, появится обновленная строка.

19. Создание автоматически создаваемых обработчиков событий

Создайте пустые обработчики событий щелчка для четырех кнопок в форме Филлорканцел, дважды щелкнув кнопки. При двойном щелчке кнопки также добавляется автоматически созданный код в файл кода конструктора, который позволяет нажать кнопку для вызова события.

20. Добавление кода для логики формы Филлорканцел

Чтобы завершить логику формы Филлорканцел, выполните следующие действия.

1. Перенесите следующие два пространства имен в область, чтобы не указывать полные имена их членов.

C#Копировать

```
using System.Data.SqlClient;
using System.Text.RegularExpressions;
```

2. Добавьте в класс переменную и вспомогательный метод, как показано в следующем коде.

C#Копировать

```
// Storage for the order ID value.
private int parsedOrderID;

/// <summary>
/// Verifies that an order ID is present and contains valid characters.
/// </summary>
private bool IsOrderIDValid()
    {
      // Check for input in the Order ID text box.
      if (txtOrderID.Text == "")
      {
        MessageBox.Show("Please specify the Order ID.");
        return false;
      }

      // Check for characters other than integers.
      else if (Regex.IsMatch(txtOrderID.Text, @"^\D*$"))
      {
        // Show message and clear input.
        MessageBox.Show("Customer ID must contain only numbers.");
        txtOrderID.Clear();
        return false;
      }
      else
      {
```

```
            // Convert the text in the text box to an integer to send to the database.
            parsedOrderID = Int32.Parse(txtOrderID.Text);
            return true;
        }
    }
```

3. Заполните основные тексты методов для четырех обработчиков событий нажатия кнопки, как показано в следующем коде.

C#Копировать

```
/// <summary>
/// Executes a t-SQL SELECT statement to obtain order data for a specified
/// order ID, then displays it in the DataGridView on the form.
/// </summary>
private void btnFindByOrderID_Click(object sender, EventArgs e)
    {
        if (IsOrderIDValid())
        {
       using (SqlConnection connection = new SqlConnection(Properties.Settings.Default.connString))
            {
                // Define a t-SQL query string that has a parameter for orderID.
                const string sql = "SELECT * FROM Sales.Orders WHERE orderID = @orderID";

                // Create a SqlCommand object.
                using (SqlCommand sqlCommand = new SqlCommand(sql, connection))
                {
                    // Define the @orderID parameter and set its value.
                    sqlCommand.Parameters.Add(new SqlParameter("@orderID", SqlDbType.Int));
                    sqlCommand.Parameters["@orderID"].Value = parsedOrderID;

                    try
                    {
                        connection.Open();

                        // Run the query by calling ExecuteReader().
                        using (SqlDataReader dataReader = sqlCommand.ExecuteReader())
                        {
                            // Create a data table to hold the retrieved data.
                            DataTable dataTable = new DataTable();

                            // Load the data from SqlDataReader into the data table.
                            dataTable.Load(dataReader);

                            // Display the data from the data table in the data grid view.
                            this.dgvCustomerOrders.DataSource = dataTable;

                            // Close the SqlDataReader.
                            dataReader.Close();
                        }
                    }
                    catch
                    {
                        MessageBox.Show("The requested order could not be loaded into the form.");
```

```
                }
                finally
                {
                    // Close the connection.
                    connection.Close();
                }
            }
        }
    }
}

/// <summary>
/// Cancels an order by calling the Sales.uspCancelOrder
/// stored procedure on the database.
/// </summary>
private void btnCancelOrder_Click(object sender, EventArgs e)
    {
        if (IsOrderIDValid())
        {
            // Create the connection.
      using (SqlConnection connection = new
SqlConnection(Properties.Settings.Default.connString))
            {
                // Create the SqlCommand object and identify it as a stored procedure.
          using (SqlCommand sqlCommand = new SqlCommand("Sales.uspCancelOrder",
connection))
                {
                    sqlCommand.CommandType = CommandType.StoredProcedure;

                    // Add the order ID input parameter for the stored procedure.
                    sqlCommand.Parameters.Add(new SqlParameter("@orderID", SqlDbType.Int));
                    sqlCommand.Parameters["@orderID"].Value = parsedOrderID;

                    try
                    {
                        // Open the connection.
                        connection.Open();

                        // Run the command to execute the stored procedure.
                        sqlCommand.ExecuteNonQuery();
                    }
                    catch
                    {
                        MessageBox.Show("The cancel operation was not completed.");
                    }
                    finally
                    {
                        // Close connection.
                        connection.Close();
                    }
                }
            }
```

```
        }
    }

/// <summary>
/// Fills an order by calling the Sales.uspFillOrder stored
/// procedure on the database.
/// </summary>
private void btnFillOrder_Click(object sender, EventArgs e)
    {
        if (IsOrderIDValid())
        {
            // Create the connection.
       using (SqlConnection connection = new
SqlConnection(Properties.Settings.Default.connString))
            {
               // Create command and identify it as a stored procedure.
          using (SqlCommand sqlCommand = new SqlCommand("Sales.uspFillOrder",
connection))
               {
                  sqlCommand.CommandType = CommandType.StoredProcedure;

                  // Add the order ID input parameter for the stored procedure.
                  sqlCommand.Parameters.Add(new SqlParameter("@orderID", SqlDbType.Int));
                  sqlCommand.Parameters["@orderID"].Value = parsedOrderID;

                  // Add the filled date input parameter for the stored procedure.
                  sqlCommand.Parameters.Add(new SqlParameter("@FilledDate",
SqlDbType.DateTime, 8));
                  sqlCommand.Parameters["@FilledDate"].Value = dtpFillDate.Value;

                  try
                  {
                     connection.Open();

                     // Execute the stored procedure.
                     sqlCommand.ExecuteNonQuery();
                  }
                  catch
                  {
                     MessageBox.Show("The fill operation was not completed.");
                  }
                  finally
                  {
                     // Close the connection.
                     connection.Close();
                  }
               }
            }
        }
    }

/// <summary>
```

```
/// Closes the form.
/// </summary>
private void btnFinishUpdates_Click(object sender, EventArgs e)
    {
        this.Close();
    }
```

Практическая работа №27 Создание запросов к БД

Для выполнения запросов к базе данных SQLite применяется класс **SqliteCommand**, который представляет реализацию интерфейса System.Data.IDbCommand. Для создания объекта **SqliteCommand** можно использовать один из его конструкторов:

SqliteCommand()

SqliteCommand(String): создает объект SqliteCommand, в конструктор которого передается выполняемое выражение SQL

SqliteCommand(String, SqliteConnection): создает объект SqliteCommand, в конструктор которого передается выполняемое выражение SQL и используемое подключение к базе данных в виде объекта SqliteConnection

SqliteCommand(String, SqliteConnection, SqliteTransaction): третий параметр представляет применяемую транзакцию в виде объекта SqliteTransaction

Альтернативным способом создания объекта SqliteCommand представляет метод CreateCommand90 класса SqliteConnection:

```
using (var connection = new SqliteConnection("Data Source=usersdata.db"))
{
connection.Open();
SqliteCommand command = connection.CreateCommand();
}
```

Для конфигурации объекта **SqliteCommand** можно использовать ряд его свойств, некоторые из них:

CommandText: хранит выполняемую команду SQL

CommandTimeout: хранит временной интервал в секундах, после которого SqliteCommand прекращает попытки выполнить команду. По умолчанию равен 30 секундам. Значение 0 представляет отстутсвие интервала.

Parameters: предствляет параметры команды

Connection: предоставляет используемое подключение SqliteConnection

Например, установим свойства подключения и выполняемой команды:

```
using (var connection = new SqliteConnection("Data Source=usersdata.db"))
{
connection.Open();
SqliteCommand command = new SqliteCommand();
command.Connection = connection;

    command.CommandText = "CREATE TABLE Users(_id INTEGER NOT NULL PRIMARY
NULL)";

}
```

Чтобы выполнить команду, необходимо применить один из методов SqliteCommand:

ExecuteNonQuery: выполняет sql-выражение и возвращает количество измененных записей. Подходит для sql-выражений INSERT, UPDATE, DELETE, CREATE.

ExecuteReader(): выполняет sql-выражение и возвращает считанные из таблицы строки. Подходит для sql-выражения SELECT.

ExecuteScalar(): выполняет sql-выражение и возвращает одно скалярное значение, например, число. Подходит для sql-выражения SELECT в паре с одной из встроенных функций SQL, как например, Min, Max, Sum, Count.

Создание таблицы

Для создания базы данных применяется SQL-команда CREATE TABLE, после которой указывается имя создаваемой таблицы и в скобках определения столбцов.

Например, создадим таблицу "Users", которая будет иметь три столбца - _id (уникальный идентификатор), Name (имя), Age (возраст):

```
using System;
using Microsoft.Data.Sqlite;
```

```
3
4       namespace HelloApp
5       {
6          class Program
7          {
8             static void Main(string[] args)
9             {
10               using (var connection = new SqliteConnection("Data Source=usersdata.db"))
11               {
12                  connection.Open();
13
14                  SqliteCommand command = new SqliteCommand(); command.Connec-
15                  tion = connection;
16                  command.CommandText = "CREATE TABLE Users(_id INTEGER NOT
17     NULL PRIMARY KEY AUTOINCREMENT UNIQUE, Name TEXT NOT NULL,
18     Age INTEGER NOT NULL)";
19                  command.ExecuteNonQuery();
20
21                  Console.WriteLine("Таблица Users создана");
22               }
23               Console.Read();
24            }
           }
        }
```

После выполнения команды в базе данных можно будет найти таблицу Users:

Для просмотра бд SQLite можно использовать специальный инструмент - DB Browser for SQLite.

Добавление данных

Теперь добавим в выше созданную таблицу Users новый объект:

```
1    using System;
2    using Microsoft.Data.Sqlite;
3
4    namespace HelloApp
5    {
6    class Program
7    {
8    static void Main(string[] args)
9    {
10   using (var connection = new SqliteConnection("Data Source=usersdata.db"))
11   {
12   connection.Open();
13
14                  SqliteCommand command = new SqliteCommand();
15                  command.Connection = connection;
16                  command.CommandText = "INSERT INTO Users (Name, Age) VALUES
17        ('Tom', 36)";
18                  int                 number                 =
command.ExecuteNonQuery(); 19
20                  Console.WriteLine($"В таблицу Users добавлено объектов: {number}");
21               }
22               Console.Read();
```

```
        }
    }
}
```

Для вставки объекта используется sql-выражение INSERT, которое имеет следующий синтаксис:

```
INSERT INTO название_таблицы (столбец1, столбец2, столбецN) VALUES (значение1, значение2, значениеN)
```

В ранее созданной таблице Users определены три столбца - _id и Age, которые хранят целое число, и Name, который хранит строку. Поэтому соответственно мы добавляем для столбца Name значение 'Tom', а для столбца Age число 36.

Здесь метод ExecuteNonQuery() возвращает число затронутых строк (в данном случае добавленных в таблицу объектов). Хотя нам необязательно возвращать результат метода, но данный результат может использоваться в качестве проверки, что операция, в частности, добавление, прошла успешно.

После добавления данных мы сможем их увидеть через DB Browser for SQLite:

Подобным образом можно добавить несколько объектов:

```
using System;
using Microsoft.Data.Sqlite;

namespace HelloApp
{
    class Program
    {
        static void Main(string[] args)
        {
            string sqlExpression = "INSERT INTO Users (Name, Age) VALUES ('Alice', 32), ('Bob', 28)";
            using (var connection = new SqliteConnection("Data Source=usersdata.db"))
            {
                connection.Open();

                SqliteCommand command = new SqliteCommand(sqlExpression, connection);

                int number = command.ExecuteNonQuery();

                Console.WriteLine($"В таблицу Users добавлено объектов: {number}");
            }
            Console.Read();
        }
    }
}
```

Обновление объектов

Для обновления применяется sql-команда UPDATE, которое имеет следующий синтаксис:

```
UPDATE название_таблицы
SET столбец1=значение1, столбец2=значение2, столбецN=значениеN
WHERE некоторый_столбец=некоторое_значение
```

Применим эту команду:

```
using System;
```

```
using Microsoft.Data.Sqlite;

namespace HelloApp
{
class Program
{
static void Main(string[] args)
{
string sqlExpression = "UPDATE Users SET Age=20 WHERE Name='Tom'";
using (var connection = new SqliteConnection("Data Source=usersdata.db"))
{
connection.Open();

                    SqliteCommand command = new SqliteCommand(sqlExpression, connection);

                    int                 number                  =
command.ExecuteNonQuery();
                    Console.WriteLine($"Обновлено объектов: {number}");
                }
                Console.Read();
            }
        }
    }
```

Здесь обновляется строка, в которой Name=Tom, то есть выше добавленный объект. Если в таблице будет несколько строк, у которых Name=Tom, то обновятся все эти строки.

Удаление

Удаление производится с помощью sql-выражения DELETE, которое имеет следующий синтаксис:

```
DELETE FROM таблица
WHERE столбец = значение
```

Удалим, например, всех пользователей, у которых имя Tom:

```
System; Microsoft.Da-
ta.Sqlite;

pace HelloApp

s Program

atic void Main(string[] args)

 string sqlExpression = "DELETE  FROM Users WHERE Name='Tom'"; us-
 ing (var connection = new SqliteConnection("Data Source=usersdata.db"))
 {
    connection.Open();

    SqliteCommand command = new SqliteCommand(sqlExpression, connection);

    int number = command.ExecuteNonQuery();

    Console.WriteLine($"Удалено объектов: {number}");
 }
 Console.Read();
```

Во всех трех случаях фактически меняется только sql-выражение, а остальная логика остается неизменной. И мы также можем выполнять сразу несколько операций:

```
using System;
using Microsoft.Data.Sqlite;

namespace HelloApp
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("Введите имя:");
string name = Console.ReadLine();

Console.WriteLine("Введите возраст:");
int age = Int32.Parse(Console.ReadLine());

string sqlExpression = $"INSERT INTO Users (Name, Age) VALUES ('{name}', {age})";
using (var connection = new SqliteConnection("Data Source=usersdata.db"))
{
connection.Open();

// добавление
SqliteCommand command = new SqliteCommand(sqlExpression, connection);
int number = command.ExecuteNonQuery();
Console.WriteLine($"Добавлено объектов: {number}");

// обновление ранее добавленного объекта
Console.WriteLine("Введите новое имя:");
name = Console.ReadLine();
sqlExpression = $"UPDATE Users SET Name='{name}' WHERE Age={age}";
command.CommandText = sqlExpression;
number = command.ExecuteNonQuery();
Console.WriteLine($"Обновлено объектов: {number}");

}
Console.Read();

}
}
```

Консольный вывод:

```
Введите имя:
Tom
Введите возраст:
36
Добавлено объектов: 1
Введите новое имя:
Sam
Обновлено объектов: 1
```

Практическая работа №28 Создание хранимых процедур

Сначала вы добавляете хранимую процедуру в ваш проект (при помощи использования меню Project и выбора пункта Add Stored Procedure). В проект будет добавлен новый класс. В листинге 18.1 показан базовый код. который имеется в файле нового класса. Вы можете добавить свой код в статическую процедуру UpdateEmployeeLogin.

\pannMvn K*31

```
using System; using System.Data; using System.Data.SalClient; using System.Data.SalTypes; using Microsoft.SalServer.Server;
public partial class StoredProcedures
[Microsoft.SqlServer.Server.SqlProcedure] public static void UpdateEmployeeLogin()
    {
// Здесь вы можете разместить свой код
    }
```

Все объекты управляемого кода (в проекте SQL Server) для выполнения своей работы используют классы данных .NET Framework (т. е. ADO.NET). Это означает, что написанные вами хранимые процедуры приведут к созданию и использованию экземпляров таких классов, как SqlConnection и SqlCommand. Код, который вы пишете, идентичен коду доступа к данным, который вы писали бы в любом другом типе проекта .NET: библиотеке классов, Web-проекте или проекте Windows-форм. Поскольку общим знаменателем является использование классов ADO.NET, то разработчикам не нужно изучать других языков (вроде T- SQL) для работы с базой данных.

Примечание

В задачи данной главы не входит рассмотрение преимуществ и недостатков написания объектов баз данных на управляемом языке по сравнению с T-SQL. Обратитесь к докладу фирмы Microsoft с названием "Using CLR Integration in SQL Server 2005", который имеется в MSDN. Несмотря на то, что он достаточно старый (написан в ноябре 2004 года), в нем хорошо изложена данная тема, и мы настоятельно рекомендуем его прочитать.

В листинге 18.2 показана процедура на языке C#, которая обновит таблицу Employee базы данных AdventureWorks информацией для входа в систему. Этот код несложен и понятен для любого, у кого есть опыт доступа к данным при помощи языка C#.

```
using System; using System.Data; using System.Data.SqlClient; using System.Data.SqlTypes; using Microsoft.SqlServer.Server;
public partial class StoredProcedures
[Microsoft.SqlServer.Server.SqlProcedure]
public static void UpdateEmployeeLogin(Sqllnt32 employeeId, Sqllnt32 managerId, SqlString loginId, SqlString title, SqlDateTime hireDate, SqlBoolean currentFlag)
using (SqlConnection conn = new SqlConnection("context connection=true"))
    {
SqlCommand UpdateEmployeeLoginCommand = new SqlCommand () ;
UpdateEmployeeLoginCommand. CommandText =
"update HumanResources.Employee SET Managerld = " + managerId.ToString() + ",
Loginld = '" + loginId.ToString() + ",M +
", Title = 1" + title.ToString() + "'" +
", HireDate = '" + hireDate.ToString() + "'" + ",
CurrentFlag = " + currentFlag.ToString() +
" WHERE Employeeld = " + employeeId.ToString();
UpdateEmployeeLoginCommand.Connection = conn;
conn.Open();
UpdateEmployeeLoginCommand.ExecuteNonQuery(); conn.Close();
    }
```

Одна строка кода заслуживает более подробного объяснения. Объект SqlConnection создается следующим образом:

SqlConnection conn = new SqlConnection("context connection=true")

Строка подключения "context connection=true" говорит движку провайдеров данных о том, что подключение должно быть создано в том же контексте, что и вызывающее прило- жение. Поскольку эта процедура будет работать внутри базы данных, то это означает, что вы будете и подключаться к базе данных хоста в контексте (транзакционном и прочем) вызывающего приложения, и работать в нем. Поэтому вам не нужно жестко прописывать здесь полностью всю строку подключения SQL.

Для сравнения в листинге 18.3 показан тот же самый запрос обновления на языке T-SQL.

```
ALTER PROCEDURE [HumanResources].[uspUpdateEmployeeLogin] @EmployeeID [int],
@ManagerID [int],
@LoginID [nvarchar](256),
@Title [nvarchar] (50),
@HireDate [datetime],
@CurrentFlag [dbo].[Flag]
WITH EXECUTE AS CALLER AS BEGIN
SET NOCOUNT ON;
BEGIN TRY
UPDATE [HumanResources].[Employee]
SET [ManagerlD] = @ManagerID ,[LoginID] = @LoginID , [Title] = @Title ,[HireDate] = @HireDate
,[CurrentFlag] = @CurrentFlag WHERE [EmployeelD] = @EmployeeID;
END TRY BEGIN CATCH
EXECUTE [dbo].[uspLogError];
END CATCH;
END;
```

МДК.01.02 Поддержка и тестирование программных модулей

Практическая работа №1

Тема: Тестирование «белым ящиком»

Цель работы: изучить методы тестирования логики программы, формализованные описания результатов тестирования и стандарты по составлению схем программ.

Теоретическая часть. Виды тестирования

Тестирование программного обеспечения включает в себя целый комплекс действий, аналогичных последовательности процессов разработки программного обеспечения. В него входят :

- постановка задачи для теста;
- проектирование теста;
- написание тестов;
- тестирование тестов;
- выполнение тестов;
- изучение результатов тестирования.

Наиболее важным является проектирование тестов. Существуют разные подходы к проектированию тестов.

Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название — стратегия «черного ящика».

Второй подход — стратегия «белого ящика», основан на анализе логики программы. При таком подходе тестирование заключается в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным. Реализация тестирования методом «черного ящика» сводится к проверке всех возможных комбинаций входных данных. Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются в программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Тестирование методом «белого ящика» также не дает 100%-ной гарантии того, что модуль не содержит ошибок. Даже если предположить, что выполнены тесты для всех ветвей алгоритма, нельзя с полной уверенностью утверждать, что программа соответствует ее спецификациям. Например, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то реализация будет совершенно неправильной, даже если проверить все пути. Вторая проблема — отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки. И наконец, проблема зависимости результатов тестирования от входных данных. Одни данные будут давать правильные результаты, а другие нет. Например, если для определения равенства трех чисел программируется выражение вида:

IF *(A* + *B*+ С)/3 = Д

то оно будет верным не для всех значений *A, B* и *C* (ошибка возникает в том случае, когда из двух значений *B* или С одно больше, а другое на столько же меньше *A).* Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно, т. е. никакое тестирование не гарантирует полное отсутствие ошибок в программе. Поэтому необходимо проектировать тесты таким образом, чтобы увеличить вероятность обнаружения ошибки в программе.

Стратегия «белого ящика»

Существуют следующие методы тестирования по принципу «белого ящика»:

- покрытие операторов;
- покрытие решений;
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.

Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

Пример.

Если для тестирования задать значения переменных *A = 2,* B = 0, *X=3,* будет реализован путь *ace,* т. е. каждый оператор программы выполнится один раз (рис. Л5.1, *а).* Но если внести в алгоритм ошибки — заменить в первом условии *and* на *or,* а во втором *X*> 1 на *X*< 1 (рис. Л5.1, *б),* ни одна ошибка не будет обнаружена (табл. Л5.1). Кроме того, путь *abd* вообще не будет охвачен тестом, и если в нем есть ошибка, она также не будет обнаружена. В табл. Л5.1 ожидаемый результат определяется по блок-схеме на рис. Л5.1, *а,* а фактический — по рис. Л5.1, *б.*

Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица Л5.1. **Результат тестирования методом покрытия** операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 2, B = 0, X=3	*X=2,5*	*X=2,5*	Неуспешно

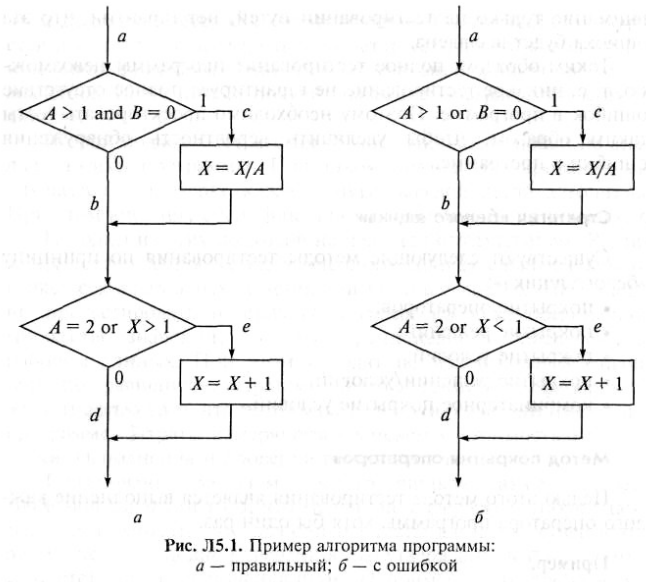


Рис. Л5.1. Пример алгоритма программы:
а — правильный; *б* — с ошибкой

Метод покрытия решений (покрытия переходов)

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз. Этот метод включает в себя критерий покрытия операторов, так как при выполнении всех направлений переходов выполнятся все операторы, находящиеся на этих направлениях.

Для программы, приведенной на рис. Л5.1, покрытие решений может быть выполнено двумя тестами, покрывающими пути *{ace, abd\,* либо *{acd, abe].* Для этого выберем следующие исходные данные: {A = 3, B=0, *X=3}* — в первом случае и *{A = 2, B=1, X=* 1} — во втором. Однако путь, где А не меняется, будет проверен с вероятностью 50%: если во втором условии вместо условия *X*> 1 записано *X*< 1, то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в табл. Л5.2.

Таблица Л5.2. Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 3, B=0, X=3	*X=1*	*X=1*	Неуспешно
A = 2, B=1,X=1	*X=1*	*X=1,5*	Успешно

Метод покрытия условий

Этот метод может дать лучшие результаты по сравнению с предыдущими. В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, .по крайней мере, один раз.

В рассматриваемом примере имеем условия: *{A*> 1, *B = 0}, {A* = 2, *X*> 1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где *A* > 1, *A* < 1, *B*= О и *B*<> 0 в точке *a* и *A* = 2, *A*<> 2, *X*> 1 и *X*< 1 в точке *b.* Тесты, удовлетворяющие критерию покрытия условий (табл. Л5.3), и соответствующие им пути:

а) *A* = 2, B=0, *X=4 ace;*

б)A= 1, *B* = 1, *X=0 abd.*

Таблица Л5.3. Результаты тестирования методом покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 2, B=0, *X*= 4	X=3	X=3	Неуспешно
A=1, B=1,X=0	X=0	*X*=1	Успешно

Метод покрытия решений/условий

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Недостатки метода:

- не всегда можно проверить все условия;
- невозможно проверить условия, которые скрыты другими условиями;
- недостаточная чувствительность к ошибкам в логических выражениях.

Так, в рассматриваемом примере два теста метода покрытия условий

а) *A = 2,* B=0, *X=4 ace;*

б)A= 1, *B=1,X=0 abd*

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие *A* > 1 будет ложным, транслятор может не проверять условия *B=0,* поскольку при любом результате условия ,B=0 результат реше- ния *((A*> 1)&(B=0)) примет значение *ложь.* То есть в варианте на рис. Л5.1 не все результаты всех условий выполнятся в процессе тестирования.

Рассмотрим реализацию того же примера на рис. Л5.2. Наиболее полное покрытие тестами в этом случае осуществляется так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути *aceg* (тест *A* = *2,* B=0, X=4), *acdfh* (тест A = 3, B= 1,X=0), *abfh* (тест A = 0, B = 0, *X*= 0), *abfi* (тест *A* = 0, B= 0, *X*= 2).

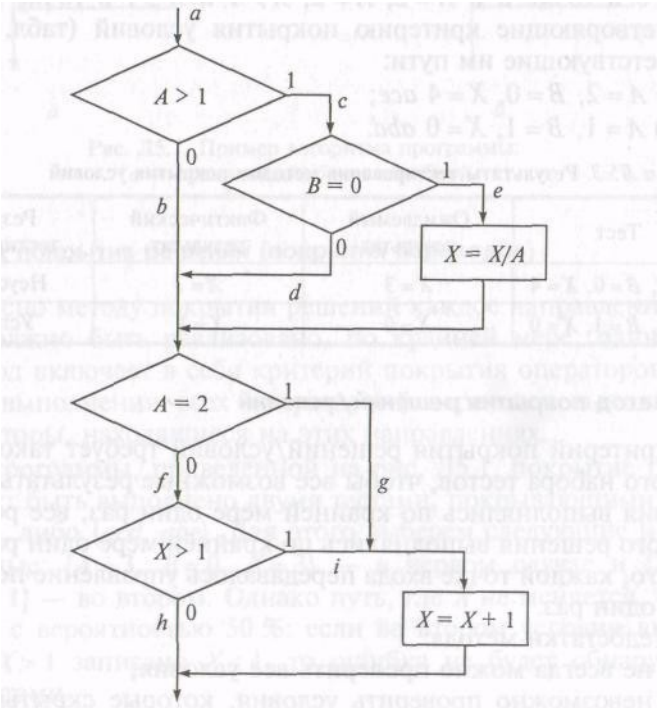


Рис. Л5.2. Пример алгоритма программы

Протестировав алгоритм на рис. Л5.2, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Метод комбинаторного покрытия условий

Критерий комбинаторного покрытия условий удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Этот метод требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

1.A>1,B=0. *5.A = 2,X>1.*

2. A > 1,B<>0. *6.A = 2,X<1.*

3.A<1,B=0. *7.A<>2,X>1.*

4.A<1,B<>0. 8.A<>2,X<1.

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами (табл. Л5.4):

A = 2, B= 0, *X*= 4 {покрывает 1, 5};

A = 2, B= 1, *X*= 1 {покрывает 2, 6};

A = 0,5, B = 0, *X*= *2* {покрывает 3, 7};

A=1,B=0, *X*= 1 {покрывает 4, 8}.

Таблица Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 2, *B=0,X=4*	*X=3*	*X=3*	Неуспешно
A= 2, *B=1,X=1*	*X=2*	*X*= 1,5	Успешно
A = 0,5, b=0,x=2	*X=3*	*X=4*	Успешно
A=1, *B=0,X*= 1	*X=1*	*X=1*	Неуспешно

Функциональное тестирование или тестирование по методом четного ящика базиру-

ется на том, что все тесты основаны на спецификации системы или ее компонентов. Поведение системы определяется изучением ее входных и соответствующих выходных данных.

При тестировании «черного ящика» тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. Например, тестирующий модуль может виртуально нажимать клавиши или кнопки мыши в тестируемой программе с помощью механизма взаимодействия процессов с уверенностью в том, что эти события вызывают тот же отклик, что и реальные нажатия клавиш и кнопок мыши.

Порядок выполнения работы

1. Спроектировать тесты по принципу «белого ящика» для программы, разработанной в лабораторной работе № 4. Использовать схемы алгоритмов, разработанные и уточненные в лабораторных работах № 2, 3.
2. Выбрать несколько алгоритмов для тестирования и обо значить буквами или цифрами ветви этих алгоритмов.
3. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования.
4. Записать тесты, которые позволят пройти по путям алгоритма.
5. Протестировать разработанную вами программу. Результаты оформить в виде таблиц (см. табл. Л5.1— Л5.4).
6. Проверить все виды тестов и сделать выводы об их эффективности.
7. Оформить отчет по лабораторной работе.
8. Сдать и защитить работу.

Защита отчета по лабораторной работе

Отчет по лабораторной работе должен состоять из:

1. Постановки задачи.
2. Блок-схемы программ.
3. Тестов.
4. Таблиц тестирования программы.
5. Выводов по результатам тестирования (не забывайте, что целью тестирования является обнаружение ошибок в программе).

Практическая работа №2

Тема: Тестирование «черным ящиком»

Методология составления тестов "чёрного ящика"

- Эквивалентное разбиение
- Анализ граничных значений
- Применение функциональных диаграмм
- Предположение об ошибке

Эквивалентное разбиение

Тестирование программы ограничивается использованием небольшого подмножества всех возможных входных данных. Тогда, конечно, хотелось бы выбрать для тестирования самое подходящее подмножество (т. с. подмножество с наивысшей вероятностью обнаружения большинства ошибок). Правильно выбранный тест этого подмножества должен обладать двумя свойствами:

уменьшать, причем более чем на единицу, число других тестов, которые должны быть разработаны для достижения заранее определенной цели "приемлемого" тестирования;

покрывать значительную часть других возможных тестов, что в некоторой степени свидетельствует о наличии или отсутствии ошибок до и после применения этого ограниченного множества значений входных данных.

Указанные свойства, несмотря на их кажущееся подобие, описывают два различных положения. Во-первых, каждый тест должен включать столько различных входных условий, сколько это возможно, с тем, чтобы минимизировать общее число необходимых тестов. Во-

вторых, необходимо пытаться разбить входную область программы на конечное число классов эквивалентности так, чтобы можно было предположить (конечно, не абсолютно уверенно), что каждый тест, являющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса. Иными словами, если один тест класса эквивалентности обнаруживает ошибку, то следует ожидать, что и все другие тесты этого класса эквивалентности будут обнаруживать ту же самую ошибку. Наоборот, если тест не обнаруживает ошибки, то следует ожидать, что ни один тест этого класса эквивалентности не будет обнаруживать ошибки (в том случае, когда некоторое подмножество класса эквивалентности не попадает в пределы любого другого класса эквивалентности, так как классы эквивалентности могут пересекаться). Эти два положения составляют основу методологии тестирования по принципу черного ящика, известной как эквивалентное разбиение. Второе положение используется для разработки набора "интересных" условий, которые должны быть протестированы, а первое - для разработки минимального набора тестов, покрывающих эти условия. Примером класса эквивалентности для программы о треугольнике является набор "трех равных чисел, имеющих целые значения, большие нуля". Определяя этот набор как класс эквивалентности, устанавливают, что если ошибка не обнаружена некоторым тестом данного набора, то маловероятно, что она будет обнаружена другим тестом набора. Иными словами, в этом случае время тестирования лучше затратить на что-нибудь другое (на тестирование других классов эквивалентности). Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

1. выделение классов эквивалентности
2. построение тестов.

Выделение классов эквивалентности

Классы эквивалентности выделяются путем выбора каждого входного условия (обычно это предложение или фраза в спецификации) и разбиением его на две или более групп.

Заметим, что различают два типа классов эквивалентности: *правильные классы*эквивалентности, представляющие правильные входные данные программы, и*неправильные классы эквивалентности*, представляющие все другие возможные состояния условий (т. е. ошибочные входные значения). При этом существует ряд правил:

1. Если входное условие описывает область значений (например, "целое данное может принимать значения от 1 до 999"), то определяются один правильный класс эквивалентности (1 <= значение целого данного <= 999) и два неправильных (значение целого данного < 1 и значение целого данного > 999).

2. Если входное условие описывает множество входных значений и есть основание полагать, что каждое значение программа трактует особо (например, "известны способы передвижения на АВТОБУСЕ, ГРУЗОВИКЕ, ТАКСИ, ПЕШКОМ или МОТОЦИКЛЕ"), то определяется правильный класс эквивалентности для каждого значения и один неправильный класс эквивалентности (например, "НА ПРИЦЕПЕ").

3. Если входное условие описывает ситуацию "должно быть" (например, "первым символом идентификатора должна быть буква"), то определяется один правильный класс эквивалентности (первый символ - буква) и один неправильный (первый символ - не буква).

4. Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности.

- Построение тестов
- Второй шаг заключается в использовании классов эквивалентности для построения тестов. Этот процесс включает в себя:

+

1. Назначение каждому классу эквивалентности уникального номера.

2. Проектирование новых тестов, каждый из которых покрывает как можно большее число непокрытых правильных классов эквивалентности, до тех пор, пока все правильные классы эквивалентности не будут покрыты (только не общими) тестами.

3. Запись тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы эквивалентности не будут покрыты тестами. Причина покрытия неправильных классов эк-

вивалентности индивидуальными тестами состоит в том, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами. Например, спецификация устанавливает "тип книги при поиске (ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА, ПРОГРАММИРОВАНИЕ или ОБЩИЙ) и количество (1-9999)". Тогда тест XYZ O отображает два ошибочных условия (неправильный тип книги и количество) и, вероятно, не будет осуществлять проверку количества, так как программа может ответить: "XYZ-НЕСУЩЕСТВУЮЩИЙ ТИП КНИГИ" и не проверять остальную часть входных данных.

- **Пример**

- Предположим, что при разработке компилятора для подмножества языка программирования требуется протестировать синтаксическую проверку оператора DIM[l].
- Пусть определена следующая спецификация:
- Оператор DIM используется для определения массивов.
- DIM *ad* [,*ad*]…,где*ad* есть описатель массива в форме*n*(*d*[,*d*]...),*n* – символическое имя массива, а*d* – индекс массива.
- Символические имена могут содержать от одного до шести символов - букв или цифр, причем первой должна быть буква.
- Допускается от одного до семи индексов. Форма индекса [lb:] ub, где lb и ub задают нижнюю и верхнюю границы индекса массива. Граница может быть либо константой, принимающей значения от - 65534 до 65535, либо целой переменной (без индексов). Если lb не определена, то предполагается, что она равна единице. Значение ub должно быть больше или равно lb. Если lb определена, то она может иметь отрицательное, нулевое или положительное значение. Оператор модет располагаться на нескольких строках (Конец спецификации.)
- Первый шаг заключается в том, чтобы идентифицировать входные условия и по ним определить классы эквивалентности. Классы эквивалентности в таблице обозначены числами.

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Число описателей массивов	ОДИН (1), > ОДНОГО (2)	НИ ОДНОГО (3)
Длина имени массива	1-6(4)	0(5), > 6(6)
Имя массива	Имеет в своем составе буквы (7) и цифры (8)	содержит что-то еще (9)
Имя массива начинается с буквы	да (10)	нет (11)
Число индексов	1-7(12)	0(13), > 7(14)
Верхняя граница	Константа (15), целая переменная (16)	имя элемента массива (17), что-то иное (18)
Имя целой переменной	Имеет в своем составе буквы (19), и цифры (20)	состоит из чего-то еще (21)

○ Целая переменная начинается с буквы	○ да (22)	○ нет (23)
○ Константа	○ От -65534 до 65535 (24))	○ Меньше -65534 (25), больше 65535 (26)
○ Нижняя граница определена	○ да (27), ○ нет (28)	○
○ Верхняя граница по отношению к нижней границе	○ Больше (29), ○ равна (30)	○ меньше (31)
○ Значение нижней границы	○ Отрицательное (32) Нуль (33), ○ > 0 (34)	○
○ Нижняя граница	○ Константа (35), ○ целая переменная	○ имя элемента массива (37), что-то
	(36)	иное (38)
○ Оператор расположен на нескольких строках	○ да (39), нет (40)	○

Следующий шаг - построение теста, покрывающего один или более правильных классов эквивалентности.

Например, тест DIM A(2) покрывает классы 1, 4, 7, 10, 12, 15, 24, 28, 29 и 40.

Далее определяются один или более тестов, покрывающих оставшиеся правильные классы эквивалентности.

Так, тест

DIM A12345(I,9,J4XXXX.65535,1,KLM, X 100),

BBB (-65534 : 100,0 : 1000,10 : 10,1 : 65535)

покрывает оставшиеся классы.

Перечислим неправильные классы эквивалентности и соответствующие им тесты:

○ (3)	○ DIMENSION
○ (5)	○ DIMENSION(10)
○ ((6)	○ DIMENSION A234567(2)
○ (9)	○ DIMENSION A.I(2)
○ (11)	○ DIMENSION1A(10)
○ (13)	○ DIMENSION B
○ (14)	○ DIMENSION B (4,4,4,4,4,4,4,4)
○ (17)	○ DIMENSION B(4,A(2))
○ (18)	○ DIMENSION B(4,,7)
○ (21)	○ DIMENSION C(1.,10)
○ (23)	○ DIMENSION C(10,1J)
○ (25)	○ DIMENSION D (-65535:1)
○ (26)	○ DIMENSION D(65536)

○ (31)	○ DIMENSION D(4:3)
○ (37)	○ DIMENSION D(A(2):4)
○ (38)	○ DIMENSION D(.:4)

Эти классы эквивалентности покрываются 18 тестами.
Хотя эквивалентное разбиение значительно лучше случайного выбора тестов, оно все же имеет недостатки (т. е. пропускает определенные типы высокоэффективных тестов).

Следующие два метода - анализ граничных значений и использование функциональных диаграмм (диаграмм причинно-следственных связей cause-effect graphing) - свободны от многих недостатков, присущих эквивалентному разбиению.

- Анализ граничных значений

Как показывает опыт, тесты, исследующие граничные условия, приносят большую пользу, чем тесты, которые их не исследуют. Граничные условия - это ситуации, возникающие непосредственно на, выше или ниже границ входных и выходных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения в двух отношениях:

1. Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.
2. При разработке тестов рассматривают не только входные условия (пространство входов), но и пространство результатов (т. е. выходные классы эквивалентности) .

Приведем несколько общих правил этого метода.
Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, если входное условие описывает область значений. Например, если правильная область входных значений есть - 1,0 - +1,0, то написать тесты для ситуаций - 1,0, 1,0, - 1,001 и 1,001.
Построить тесты для минимального и максимального значений условий и тесты, большие и меньшие этих значений, если входное условие удовлетворяет дискретному ряду значений. Например, если входной файл может содержать от 1 до 255 записей, то получить тесты для 0,1, 255 и 256 записей.
Использовать правило 1 для каждого выходного условия. Например, если программа вычисляет ежемесячный расход и если минимум расхода составляет 0,00 дол., а максимум - 1165,25 дол., то построить тесты, которые вызывают расходы с 0,00 дол. и 1165,25 дол. Кроме того, построить, если это возможно, тесты, которые вызывают отрицательный расход и расход больше 1165,25 дол. Заметим, что важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей (например, при рассмотрении подпрограммы вычисления синуса). Не всегда также можно получить результат вне выходной области, но, тем не менее, стоит рассмотреть эту возможность.
Использовать правило 2 для каждого выходного условия. Например, если система информационного поиска отображает на экране терминала наиболее релевантные рефераты в зависимости от входного запроса, но никак не более четырех рефератов, то построить тесты, такие, чтобы программа отображала нуль, один и четыре реферата, и тест, который мог бы вызвать выполнение программы с ошибочным отображением пяти рефератов.
Если вход или выход программы есть упорядоченное множество (например, последовательный файл, линейный список, таблица), то сосредоточить внимание на первом и последнем элементах этого множества.
Существенное различие между анализом граничных значений и эквивалентным разбиением заключается в том, что анализ граничных значений исследует ситуа-

ции, возникающие на и вблизи границ эквивалентных разбиений.

Пример

Положим, что нужно протестировать программу бинарного поиска. Нам известна *спецификация* этой программы. Поиск выполняется в массиве элементов *M,* возвращается индекс *I* элемента массива, значение которого соответствует ключу поиска *Key.*

Входные условия:

1) массив должен быть упорядочен;
2) массив должен иметь не менее одного элемента;
3) нижняя граница массива (индекс) должна быть меньше или равна его верхней границе.

Результаты::

1) если элемент найден, то флаг Result=True, значение I — номер элемента;
2) если элемент не найден, то флаг Result=False, значение I не определено.

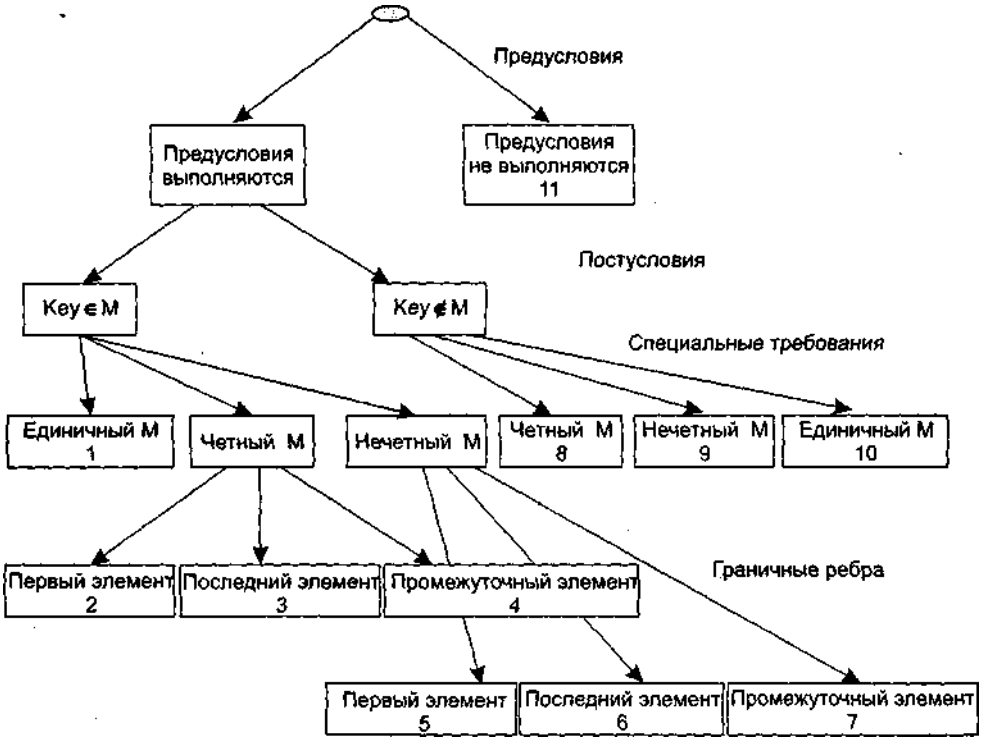
Для формирования классов эквивалентности (и их ребер) надо построить дерево разбиений. Листья дерева разбиений дадут нам искомые классы эквивалентности. Определим стратегию разбиения. На первом уровне будем анализировать выполнимость входных условий, на втором уровне — выполнимость результатов. На третьем уровне можно анализировать специальные требования, полученные из практики разработчика. В нашем примере мы знаем, что входной массив должен быть упорядочен. Обработка упорядоченных наборов из четного и нечетного количества элементов может выполняться по-разному. Кроме того, принято выделять специальный случай одноэлементного массива. Следовательно, на уровне специальных требований возможны следующие эквивалентные разбиения:

1) массив из одного элемента;
2) массив из четного количества элементов;
3) массив из нечетного количества элементов, большего единицы.

Наконец на последнем, 4-м уровне критерием разбиения может быть анализ ребер классов эквивалентности. Очевидно, возможны следующие варианты:

1) работа с первым элементом массива;
2) работа с последним элементом массива;
3) работа с промежуточным (ни с первым, ни с последним) элементом массива.

Структура дерева разбиений приведена на рис. 7.3.



Дерево разбиений области исходных данных бинарного поиска

Это дерево имеет 11 листьев. Каждый лист задает отдельный тестовый вариант.

Покажем тестовые варианты, основанные на проведенных разбиениях.

*Тестовый вариант 1 (единичный массив, элемент найден)***:**
ИД: M=15; Key=15.
ОЖ.РЕЗ.: Resutt=True; I=1.
*Тестовый вариант 2 (четный массив, найден 1-й элемент)***:**
ИД: M=15, 20, 25,30,35,40; Key=15.
ОЖ.РЕЗ.: Result=True; I=1.
Тестовый вариант 3 (четный массив, найден последний элемент) **:**
ИД: M=15, 20, 25, 30, 35, 40; Key=40.
ОЖ.РЕЗ:. Result=True; I=6.
*Тестовый вариант 4 (четный массив, найден промежуточный элемент)***:**
ИД: M=15,20,25,30,35,40; Key=25.
ОЖ.РЕЗ.: Result-True; I=3.
*Тестовый вариант 5 (нечетный массив, найден 1-й элемент)***:**
ИД: M=15, 20, 25, 30, 35,40, 45; Key=15.
ОЖ.РЕЗ.: Result=True; I=1.
*Тестовый вариант 6 (нечетный массив, найден последний элемент)***:**
ИД: M=15, 20, 25, 30,35, 40,45; Key=45.
ОЖ.РЕЗ.: Result=True; I=7.
*Тестовый вариант 7 (нечетный массив, найден промежуточный элемент)***:**
ИД: M=15, 20, 25, 30,35, 40, 45; Key=30.
ОЖ.РЕЗ.: Result=True; I=4.
*Тестовый вариант 8 (четный массив, не найден элемент)***:**
ИД: M=15, 20, 25, 30, 35,40; Key=23.
ОЖ.РЕЗ.: Result=False; I=?
*Тестовый вариант 9 (нечетный массив, не найден элемент)***;**
ИД: M=15, 20, 25, 30, 35, 40, 45; Key=24.
ОЖ.РЕЗ:. Result=False; I=?
*Тестовый вариант 10 (единичный массив, не найден элемент)***:**
ИД: M=15; Key=0.
ОЖ.РЕЗ.: Result=False; I=?
*Тестовый вариант 11 (нарушены предусловия)***:**
ИД: M=15, 10, 5, 25, 20, 40, 35; Key=35.
ОЖ.РЕЗ.: Аварийное завершение: Массив не упорядочен.

Анализ граничных значений, если он применен правильно, является одним из наиболее полезных методов проектирования тестов. Однако он часто оказывается неэффективным из-за того, что внешне выглядит простым. Граничные условия могут быть едва уловимы и, следовательно, определение их связано с большими трудностями.

Порядок выполнения работы

1. Спроектировать тесты по принципу «черного ящика» для программы, разработанной в лабораторной работе № 4. Использовать схемы алгоритмов, разработанные и уточненные в лабораторных работах № 2, 3.
2. Выбрать несколько алгоритмов для тестирования и обо значить буквами или цифрами ветви этих алгоритмов.
3. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования.
4. Записать тесты, которые позволят пройти по путям алгоритма.
5. Протестировать разработанную вами программу. Результаты оформить в виде таблиц (см. табл. Л5.1— Л5.4).

6. Проверить все виды тестов и сделать выводы об их эффективности.
7. Оформить отчет по лабораторной работе.
8. Сдать и защитить работу.

Защита отчета по лабораторной работе

Отчет по лабораторной работе должен состоять из:

1. Постановки задачи.
2. Блок-схемы программ.
3. Тестов.
4. Таблиц тестирования программы.
5. Выводов по результатам тестирования (не забывайте, что целью тестирования является обнаружение ошибок в программе).

Практическая работа №3

Тема: Модульное тестирование

Цель работы: Изучить технологии модульного тестирования. Получить практические навыки по работе с Unit Testing Framework от Microsoft.

6.2 Краткие теоретические сведения

Модульное тестирование, или unit-тестирование (англ. unit testing) процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

Как и любая технология тестирования, модульное тестирование не позволяет отловить все ошибки программы. В самом деле, это следует из практической невозможности трассировки всех возможных путей выполнения программы, за исключением простейших случаев. Кроме того, происходит тестирование каждого из модулей по отдельности. Это означает, что ошибки интеграции, системного уровня, функций, исполняемых в нескольких модулях не будут определены. Кроме того, данная технология бесполезна для проведения тестов на производительность. Таким образом, модульное тестирование более эффективно при использовании в сочетании с другими методиками тестирования.

Для получения выгоды от модульного тестирования требуется строго следовать технологии тестирования во всём процессе разработки программного обеспечения. Нужно хранить не только записи обо всех проведённых тестах, но и обо всех изменениях исходного кода во всех модулях. С этой целью следует использовать систему контроля версий ПО. Таким образом, если более поздняя версия ПО не проходит тест, который был успешно пройден ранее, будет несложным сверить исходный код вариантов и устранить ошибку. Также необходимо убедиться в неизменном отслеживании и анализе неудачных тестов. Игнорирование этого требования приведёт к лавинообразному увеличению неудачных тестовых результатов.

На данный момент наиболее распространенными инструментами модульного тестирования платформы .NET являются библиотека NUnit и Microsoft Unit Testing Framework.

Visual Studio Unit Testing Framework – инструмент модульного тестирования для платформы .NET, встроенный в среду разработки Visual Studio 2005 и выше. Чтобы определить, что класс является тестирующим, необходимо пометить его атрибутом [TestClass]. Если класс помечен этим атрибутом, то он может содержать в себе тестовые методы. Обычно тестирующий класс называют так же, как и тестируемый, только с префиксом Test. В тестирующем классе могут содержаться тестирующие методы и обычно для всех методов тестируемого класса, которые возвращают значение, создается

отдельный тестирующий метод. Тестирующий метод обычно называют, так же как и тестируемый, только с префиксом Test.

Кроме тестирующих методов в тестирующем классе могут быть методы инициализации и очистки. Метод инициализации помечается атрибутом [TestInitialize] и позволяет инициализировать необходимые переменные перед выполнением метода-теста. Метод очистки помечается атрибутом [TestCleanup] и позволяет очистить результаты выполнения теста, например, очистить файл, удалить лишние записи с базы данных, присвоить переменным значения по умолчанию.

Кроме методов инициализации и очистки на уровне теста, в тестирующем классе могут присутствовать методы инициализации и очистки уровня класса. Эти методы вызываются один раз. Методы инициализации уровня класса вызывается один раз перед вызовом первого теста, а метод очистки уровня класса вызывается после выполнения последнего теста. Метод инициализации уровня класса помечается атрибутом [ClassInitialize], а метод очистки уровня класса помечается атрибутом [ClassCleanup].

Выполнение работы (Текст программы)

```
TestGenericDAO using System;
using Microsoft.VisualStudio.TestTools.UnitTesting; using
NHibernate;
using NHibernate.Criterion;
using System.Collections.Generic; using
FluentNHibernate;
using FluentNHibernate.Cfg; using
FluentNHibernate.Cfg.Db;
using FluentNHibernate.Automapping; using
NHibernate.Tool.hbm2ddl; using
System.Reflection;
using FluentNHibernate.Mapping; using
NHibernate.Cfg;
using lab6.mapping; using
lab6.dao; using lab6.domain;
namespace lab6
{
[TestClass()]
public abstract class TestGenericDAO<T> where T : EntityBase
{
protected static ISessionFactory factory;
protected static ISession session; protected
DaoFactory daoFactory;
protected TestContext testContextInstance;
/** DAO that will be tested */ protected
IGenericDAO<T> dao = null;
/** First entity that will be used in tests */
protected T entity1 = null;
/** Second entity that will be used in tests */
protected T entity2 = null;
/** Third entity that will be used in tests */
protected T entity3 = null;
public TestGenericDAO()
{
session = openSession("localhost", 5432, "market",
"postgres", "06031992");
}
public TestContext TestContext
{
get
{
```

```
return testContextInstance;
}
set
{
testContextInstance = value;
}
}
/*Getting dao this test case works with*/ public
IGenericDAO<T> getDAO()
{
return dao;
}
/*Setting dao this test case will work with*/ public
void setDAO(IGenericDAO<T> dao)
{
this.dao = dao;
}
[ClassCleanup]
public static void ClassCleanup()
{
session.Close();
}
[TestInitialize]
public void TestInitialize()
{
Assert.IsNotNull(dao,
"Please, provide IGenericDAO implementation in constructor");
createEntities();
Assert.IsNotNull(entity1, "Please, create object for entity1");
Assert.IsNotNull(entity2, "Please, create object for entity2");
Assert.IsNotNull(entity3, "Please, create object for entity3");
checkAllPropertiesDiffer(entity1, entity2);
checkAllPropertiesDiffer(entity1, entity3);
checkAllPropertiesDiffer(entity2, entity3); saveEntitiesGeneric();
}
[TestCleanup]
public void TestCleanup()
{
try
{
if ((entity1 = dao.GetById(entity1.Id)) != null)
dao.Delete(entity1);
}
catch (Exception)
{
Assert.Fail("Problem in cleanup method");
}
try
{
if ((entity2 = dao.GetById(entity2.Id)) != null)
dao.Delete(entity2);
}
catch (Exception)
{
Assert.Fail("Problem in cleanup method");
```

```
}
try
{
if ((entity3 = dao.GetById(entity3.Id)) != null)
dao.Delete(entity3);
}
catch (Exception)
{
Assert.Fail("Problem in cleanup method");
}
entity1 = null; entity2
= null; entity3 = null;
}
[TestMethod]
public void TestGetByIdGeneric()
{
T foundObject = null;
// Should not find with inexistent id try
{
long id = DateTime.Now.ToFileTime();
foundObject = dao.GetById(id);
Assert.IsNull(foundObject, "Should return null if id is inexistent");
}
catch (Exception)
{
Assert.Fail("Should return null if object not found");
}
// Getting all three entities
getEntityGeneric(entity1.Id, entity1);
getEntityGeneric(entity2.Id, entity2);
getEntityGeneric(entity3.Id, entity3);
}
[TestMethod]
public void TestGetAllGeneric()
{
List<T> list = getListOfAllEntities();
Assert.IsTrue(list.Contains(entity1),
"After dao method GetAll list should contain entity1");
Assert.IsTrue(list.Contains(entity2),
"After dao method GetAll list should contain entity2");
Assert.IsTrue(list.Contains(entity3),
"After dao method GetAll list should contain entity3");
}
[TestMethod]
public void TestDeleteGeneric()
{
try
{
dao.Delete((T)null);
Assert.Fail("Should not delete entity will null id");
```

```
}
catch (Exception)
{
}
// Deleting second entity try
{
dao.Delete(entity2);
}
catch (Exception)
{
Assert.Fail("Deletion should be successful of entity2");
}
// Checking if other two entities can be still found
getEntityGeneric(entity1.Id, entity1);
getEntityGeneric(entity3.Id, entity3);
// Checking if entity2 can not be found try
{
T foundEntity = null;
foundEntity = dao.GetById(entity2.Id);
Assert.IsNull(foundEntity,
"After deletion entity should not be found with id " + entity2.Id);
}
catch (Exception)
{
Assert.Fail("Should return null if finding the deleted entity");
}
// Checking if other two entities can still be found in getAll list List<T>
list = getListOfAllEntities(); Assert.IsTrue(list.Contains(entity1),
"After dao method GetAll list should contain entity1");
Assert.IsTrue(list.Contains(entity3),
"After dao method GetAll list should contain entity3");
}
protected abstract void createEntities();
protected abstract void checkAllPropertiesDiffer(T entityToCheck1, T
entityToCheck2);
protected abstract void checkAllPropertiesEqual(T entityToCheck1, T
entityToCheck2);
protected void saveEntitiesGeneric()
{
T savedObject = null; try
{
dao.SaveOrUpdate(entity1);
savedObject = getPersistentObject(entity1);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity1);
entity1 = savedObject;
}
```

```
catch (Exception)
{
Assert.Fail("Fail to save entity1");
}
try
{
dao.SaveOrUpdate(entity2);
savedObject = getPersistentObject(entity2);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity2);
entity2 = savedObject;
}
catch (Exception)
{
Assert.Fail("Fail to save entity2");
}
try
{
dao.SaveOrUpdate(entity3);
savedObject = getPersistentObject(entity3);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity3);
}
catch (Exception)
{
Assert.Fail("Fail to save entity3");
}
}
protected T getPersistentObject(T nonPersistentObject)
{
        ICriteria criteria =
    session.CreateCriteria(typeof(T)).Add(Example.Create(nonPersistentObject));
IList<T> list = criteria.List<T>();
Assert.IsTrue(list.Count >= 1,
"Count of grups must be equal or more than 1"); return
list[0];
}
protected void getEntityGeneric(long id, T entity)
{
T foundEntity = null; try
{
foundEntity = dao.GetById(id);
Assert.IsNotNull(foundEntity,
"Service method getEntity should return entity if successfull");
checkAllPropertiesEqual(foundEntity, entity);
}
catch (Exception)
{
Assert.Fail("Failed to get entity with id " + id);
```

```
}
}
protected List<T> getListOfAllEntities()
{
List<T> list = null;
// Should get not null and not empty list try
{
list = dao.GetAll();
}
catch (Exception)
{
Assert.Fail(
"Should be able to get all entities that were added before");
}
Assert.IsNotNull(list,
"DAO method GetAll should return list of entities if successfull");
Assert.IsFalse(list.Count == 0,
"DAO method should return not empty list if successfull"); return
list;
}
//Метод открытия сессии
public static ISession openSession(String host, int port, String
database, String user, String passwd)
{
ISession session = null; if
(factory == null)
{
FluentConfiguration configuration = Fluently.Configure()
.Database(PostgreSQLConfiguration
.PostgreSQL82.ConnectionString(c => c
.Host(host)
.Port(port)
.Database(database)
.Username(user)
.Password(passwd)))
.Mappings(m => m.FluentMappings.Add<TovarsMap>().Add<ProdMap>())
.ExposeConfiguration(BuildSchema);
factory = configuration.BuildSessionFactory();
}
//Открытие сессии
session = factory.OpenSession(); return
session;
}
//Метод для автоматического создания таблиц в базе данных private
static void BuildSchema(Configuration config)
{
new SchemaExport(config).Create(false, true);
}
}
}
```

Текст TestProdDAO

```
using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting; using
lab6.domain;
using lab6.dao;
namespace lab6
{
[TestClass]
public class TestProdDAO : TestGenericDAO<Prod>
{
protected IProdDAO prodDAO = null;
protected Tovars tovar1 = null; protected
Tovars tovar2 = null; protected Tovars tovar3 =
null;
public TestProdDAO()
: base()
{
DaoFactory daoFactory = new NHibernateDAOFactory(session);
prodDAO = daoFactory.getProdDAO();
setDAO(prodDAO);
}
protected override void createEntities()
{
entity1 = new Prod(); entity1.NameProd =
"1";
entity1.Surname = "1";
entity1.God = 1992; entity2 =
new Prod(); entity2.NameProd =
"2";
entity2.Surname = "2";
entity2.God = 1993; entity3 =
new Prod(); entity3.NameProd =
"3";
entity3.Surname = "3";
entity3.God = 1994;
}
protected override void checkAllPropertiesDiffer(Prod entityToCheck1, Prod
entityToCheck2)
{
        Assert.AreNotEqual(entityToCheck1.NameProd, entityToCheck2.NameProd, "Values
    must be different");
        Assert.AreNotEqual(entityToCheck1.Surname,entityToCheck2.Surname, "Values must be
    different");
        Assert.AreNotEqual(entityToCheck1.God,entityToCheck2.God, "Values must be
    different");
}
protected override void checkAllPropertiesEqual(Prod entityToCheck1, Prod
entityToCheck2)
{
Assert.AreEqual(entityToCheck1.NameProd, entityToCheck2.NameProd, "Values
must be equal");
Assert.AreEqual(entityToCheck1.Surname, entityToCheck2.Surname,
```

```
"Values must be equal"); Assert.AreEqual(entityToCheck1.God,
entityToCheck2.God, "Values must be equal");
}
[TestMethod]
public void TestGetByIdProd()
{
base.TestGetByIdGeneric();
}
[TestMethod]
public void TestGetAllProd()
{
base.TestGetAllGeneric();
}
[TestMethod]
public void TestDeleteProd()
{
base.TestDeleteGeneric();
}
// [TestMethod]
// public void TestGetProdByName()
// {
// Prod prod1 = prodDAO.getProdByName(entity1.NameProd);
// Assert.IsNotNull(prod1,
// "Service method getGroupByName should return group if successfull");
// Prod prod2 = prodDAO.getProdByName(entity2.NameProd);
// Assert.IsNotNull(prod2,
// "Service method getGroupByName should return group if successfull");
// Prod prod3 = prodDAO.getProdByName(entity3.NameProd);
// Assert.IsNotNull(prod3,
// "Service method getGroupByName should return group if successfull");
// checkAllPropertiesEqual(prod1, entity1);
// checkAllPropertiesEqual(prod2, entity2);
// checkAllPropertiesEqual(prod3, entity3);
// } [TestMethod]
public void TestGetAllTovarsOfProd()
{
createEntitiesForTovars();
Assert.IsNotNull(tovar1, "Please, create object for student1");
Assert.IsNotNull(tovar2, "Please, create object for student2");
Assert.IsNotNull(tovar3, "Please, create object for student3");
entity1.TovarsList.Add(tovar1);
tovar1.Prod = entity1;
entity1.TovarsList.Add(tovar2);
tovar2.Prod = entity1;
entity1.TovarsList.Add(tovar3);
tovar3.Prod = entity1;
Prod savedObject = null; try
{
dao.SaveOrUpdate(entity1);
```

```
savedObject = getPersistentObject(entity1);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity1);
entity1 = savedObject;
}
catch (Exception)
{
Assert.Fail("Fail to save entity1");
}
IList<Tovars> tovarsList =
prodDAO.getAllTovarsOfProd(entity1.NameProd);
Assert.IsNotNull(tovarsList, "List can't be null");
Assert.IsTrue(tovarsList.Count == 3,
"Count of students in the list must be 3");
checkAllPropertiesEqualForTovars(tovarsList[0], tovar1);
checkAllPropertiesEqualForTovars(tovarsList[1], tovar2);
checkAllPropertiesEqualForTovars(tovarsList[2], tovar3);
}
//[TestMethod]
//public void TestDelProdByName()
//{
// try
// {
// prodDAO.delProdByName(entity2.NameProd);
// }
// catch (Exception)
// {
// Assert.Fail("Deletion should be successful of entity2");
// }
// Checking if other two entities can be still found
// getEntityGeneric(entity1.Id, entity1);
// getEntityGeneric(entity3.Id, entity3);
// Checking if entity2 can not be found
// try
// {
// Prod foundProd = null;
// foundProd = dao.GetById(entity2.Id);
// Assert.IsNull(foundProd,
// "After deletion entity should not be found with groupName " +
// entity2.NameProd);
// }
// catch (Exception)
// {
// Assert.Fail("Should return null if finding the deleted entity");
// }
// Checking if other two entities can still be found in getAll list
// List<Prod> list = getListOfAllEntities();
// Assert.IsTrue(list.Contains(entity1),
// "After dao method GetAll list should contain entity1");
// Assert.IsTrue(list.Contains(entity3),
// "After dao method GetAll list should contain entity3");
```

```
// }
protected void createEntitiesForTovars()
{
tovar1 = new Tovars();
tovar1.Name = "C";
tovar1.Price = 10;
tovar1.Ves = 100; tovar2 = new
Tovars(); tovar2.Name = "B";
tovar2.Price = 20;
tovar2.Ves = 200; tovar3 = new
Tovars(); tovar3.Name = "D";
tovar3.Price = 30;
tovar3.Ves = 300;
}
protected void checkAllPropertiesEqualForTovars(Tovars entityToCheck1, Tovars
entityToCheck2)
{
Assert.AreEqual(entityToCheck1.Name, entityToCheck2.Name, "Values
must be equal"); Assert.AreEqual(entityToCheck1.Price,
entityToCheck2.Price, "Values must be equal");
Assert.AreEqual(entityToCheck1.Ves, entityToCheck2.Ves, "Values
must be equal");
}
}
}
```

Текст TestTovarsDAO using

```
System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting; using
lab6.domain;
using lab6.dao;
using NHibernate.Criterion; using
NHibernate; namespace lab6
{
[TestClass]
public class TestTovarsDAO : TestGenericDAO<Tovars>
{
protected ITovarsDao tovarsDAO = null;
protected IProdDAO prodDAO = null; protected
Prod prod = null;
public TestTovarsDAO()
: base()
{
DaoFactory daoFactory = new NHibernateDAOFactory(session);
tovarsDAO = daoFactory.getTovarsDAO();
prodDAO = daoFactory.getProdDAO(); setDAO(tovarsDAO);
}
```

```
protected override void createEntities()
{
entity1 = new Tovars();
entity1.Name = "C";
entity1.Price = 10;
entity1.Ves = 100; entity2 =
new Tovars(); entity2.Name =
"B"; entity2.Price = 20;
entity2.Ves = 200; entity3 =
new Tovars(); entity3.Name =
"D"; entity3.Price = 30;
entity3.Ves = 300;
}
protected override void checkAllPropertiesDiffer(Tovars entityToCheck1, Tovars
entityToCheck2)
{
Assert.AreNotEqual(entityToCheck1.Name, entityToCheck2.Name, "Values
must be different"); Assert.AreNotEqual(entityToCheck1.Price,
entityToCheck2.Price, "Values must be different");
Assert.AreNotEqual(entityToCheck1.Ves, entityToCheck2.Ves, "Values
must be different");
}
protected override void checkAllPropertiesEqual(Tovars entityToCheck1, Tovars
entityToCheck2)
{
Assert.AreEqual(entityToCheck1.Name, entityToCheck2.Name, "Values
must be equal"); Assert.AreEqual(entityToCheck1.Price,
entityToCheck2.Price, "Values must be equal");
Assert.AreEqual(entityToCheck1.Ves, entityToCheck2.Ves, "Values
must be equal");
}
[TestMethod]
public void TestGetByIdTovars()
{
base.TestGetByIdGeneric();
}
[TestMethod]
public void TestGetAllTovars()
{
base.TestGetAllGeneric();
}
[TestMethod]
public void TestDeleteTovars()
{
base.TestDeleteGeneric();
}
[TestMethod]
public void TestGetTovarByProdName()
```

```
{
Prod prod = new Prod();
prod.NameProd = "1";
prod.Surname = "1";
prod.God = 1992;
prod.TovarsList.Add(entity1); entity1.Prod
= prod; prod.TovarsList.Add(entity2);
entity2.Prod = prod;
prod.TovarsList.Add(entity3); entity3.Prod
= prod;
Prod savedProd = null; try
{
prodDAO.SaveOrUpdate(prod); savedProd =
getPersistentProd(prod);
Assert.IsNotNull(savedProd,
"DAO method saveOrUpdate should return group if successfull");
checkAllPropertiesEqualProd(savedProd, prod);
prod = savedProd;
}
catch (Exception)
{
Assert.Fail("Fail to save group");
}
getTovarByProdName(entity1, prod.NameProd,
entity1.Name);
getTovarByProdName(entity2, prod.NameProd,
entity2.Name);
getTovarByProdName(entity3, prod.NameProd,
entity3.Name); prod.TovarsList.Remove(entity1);
prod.TovarsList.Remove(entity2);
prod.TovarsList.Remove(entity3);
entity1.Prod = null; entity2.Prod
= null; entity3.Prod = null;
prodDAO.Delete(prod);
}
protected void getTovarByProdName(Tovars tovar, string nameProd, string name)
{
Tovars foundTovars = null; try
{
foundTovars = tovarsDAO.getTovarByProdName(
nameProd,name);
Assert.IsNotNull(tovarsDAO,
"Service method should return student if successfull");
checkAllPropertiesEqual(foundTovars, tovar);
}
catch (Exception)
{
```

```
Assert.Fail("Failed to get student with nameProd " + nameProd +
" name " + name);
}
}
protected Prod getPersistentProd(Prod nonPersistentProd)
{
ICriteria criteria = session.CreateCriteria(typeof(Prod))
.Add(Example.Create(nonPersistentProd)); IList<Prod> list =
criteria.List<Prod>(); Assert.IsTrue(list.Count >= 1,
"Count of grups must be equal or more than 1"); return
list[0];
}
protected void checkAllPropertiesEqualProd(Prod entityToCheck1, Prod
entityToCheck2)
{
Assert.AreEqual(entityToCheck1.NameProd, entityToCheck2.NameProd, "Values
must be equal");
Assert.AreEqual(entityToCheck1.Surname, entityToCheck2.Surname, "Values
must be equal");
Assert.AreEqual(entityToCheck1.God, entityToCheck2.God,
+"Values must be equal");
}
}
}
```

Выводы

В ходе выполнения данной лабораторной работы были изучены технологии модульного тестирования. Были получены практические навыки по работе с Unit Testing Framework от Microsoft.

Модульное тестирование, или unit-тестирование (англ. unit testing) процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Практическая работа №4

Тема: Интеграционное тестирование

Цель работы: Овладение навыками интеграционного тестирования.

Общие сведения

Интеграционное тестирование называют еще тестированием архитектуры системы. С одной стороны, это название обусловлено тем, что интеграционные тесты включают в себя проверки всех возможных видов взаимодействий между программными модулями и элементами, которые определяются в архитектуре системы - таким образом, интеграционные тесты проверяют полноту взаимодействий в тестируемой реализации системы. С другой стороны, результаты выполнения интеграционных тестов - один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е., с этой точки зрения, интеграционные тесты проверяют корректность взаимодействия компонент системы.

В результате проведения интеграционного тестирования и устранения всех выявленных дефектов получается согласованная и целостная архитектура программной системы, т.е. можно считать, что интеграционное тестирование - это тестирование архитектуры и низкоуровневых функциональных требований. Интеграционное тестирование, как правило, представляет собой итеративный процесс, при котором проверяется совокупность модулей,

возрастающая от итерации к итерации. В интеграционном тестирование выделяют три методы выполнения: восходящее тестирование; монолитное тестирование; нисходящее тестирование.

2. **Задание**

Согласно варианту провести один из методов интеграционного тестирования.

Практическая работа №5

Тема: Оформление документации на программные средства с использованием инструментальных средств.

Цель работы Овладение навыками документирования результатов тестирования. **Общие сведения**

Каждый дефект, обнаруженный в процессе тестирования, должен быть задокументирован и отслежен. При обнаружении нового дефекта его заносят в базу дефектов. При занесении нового дефекта рекомендуется указывать, как минимум, следующую информацию:

1) Наименование подсистемы, в которой обнаружен дефект.
2) Версия продукта (номер build), на котором дефект был найден.
3) Описание дефекта.
4) Описание процедуры (шагов, необходимых для воспроизведения дефекта).
5) Номер теста, на котором дефект был обнаружен.
6) Уровень дефекта, то есть степень его серьезности с точки зрения критериев качества продукта или заказчика.

Тестовый отчет обновляется после каждого цикла тестирования и должен содержать следующую информацию для каждого цикла:

Задание

Задокументировать результаты тестирования.

МДК.01.03 Разработка мобильных приложений

Практическая работа № 1.

Тема:Платформа Android

Цель работы: Изучить основы работы с платформой Android Начало работы с Android

Бурное развитие информационных технологий в последнее время привело к тому, что появилось много новых устройств и технологий, таких, как планшеты, смартфоны, нетбуки, другие гаджеты. Они все более прочно входят в нашу жизнь и становятся привычным делом. Лидирующей платформой среди подобных гаджетов на сегодняшний день является ОС Андроид.

Android используется на самых разных устройствах. Это и смартфоны, и планшеты, и телевизоры, и смарт-часы и ряд других гаджетов. По разным подсчетам за 2020 год этой операционой системой пользуются около 85% владельцев смартфонов, а общее количество пользователей сматрфонов на ОС Android оценивается в более чем 2,5 млрд. человек по всему миру.

ОС Андроид была создана разработчиком Энди Рубином (Andy Rubin) в качестве операционной системы для мобильных телефонов и поначалу развивалась в рамках компании Android Inc. Но в 2005 году Google покупает Android Inc. и начинает развивать операционную систему с новой силой. Android постоянно эволюционирует, и вместе с операционной системой эволюционируют средства и инструменты для разработки. На данный момент (ноябрь 2020 года) последней версией является Android 11.0, которая вышла в сентябре 2020 года:

Таблица 1

Версия	**Кодовое имя**	**Дата выпуска**	**Уровень API**
11.0	11	8 сентября 2020	30
10.0	10	3 сентября 2019	29
9.0	Pie	6 августа 2018	28

8.1	Oreo	5 декабря 2017	27
8.0	Oreo	21 августа 2017	26
7.1	Nougat	4 октября 2016	25
7.0	Nougat	22 августа 2016	24
6.0	Marshmallow	5 октября 2015	23
5.1	Lollipop	9 марта 2015	22
5.0	Lollipop	3 ноября 2014	21
4.4	KitKat	31 октября 2013	19
4.3	Jelly Bean	24 июля 2013	18
4.2	Jelly Bean	13 ноября 2012	17
4.1	Jelly Bean	9 июля 2012	16
4.0	Ice Cream Sandwich	16 декабря 2011	15
2.3	Gingerbread	6 декабря 2010	10

Что нужно для разработки?

Стоит отметить, что разрабатывать приложения под Android можно с помощью различных фреймворков и языков программирования. Так, в качестве языков программирования могут применяться Java, Kotlin, Dart (фреймворк Flutter), C++, Python, C# (платформа Xamarin) и т.д. В данном руководстве мы будем использовать именно язык Java, как наиболее распространенный и используемый. Поэтому прежде чем приступать к освоению программирования под Android по данному руководству, необходимо освоить хотя бы базовые момент языка Java.

Установка средств разработки

Существуют разные среды разработки для Android. Рекомендуемой средой разработки является Android Studio, которая создана специально для разработки под ОС Android. Поэтому мы ее и будем использовать. Загрузить файл установщика можно с официального сайта: https://developer.android.com/studio:

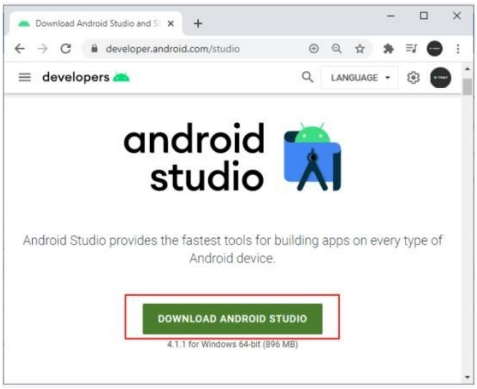


Рисунок 1

Кроме самой среды Android Studio для разработки также потребуется набор инструментов, который называется Android SDK. Например, если ранее Android SDK еще не было установлено, то при первом обращении к Android Studio она сообщит, что Android SDK отсутствует.

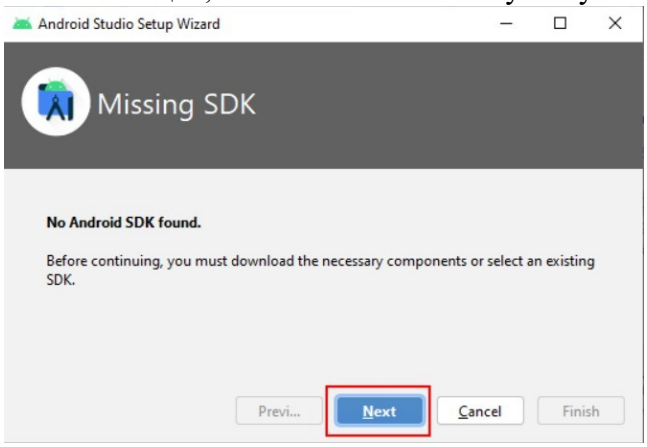


Рисунок 2

Мы можем отдельно вручную загрузить Android SDK с официального сайта и установить его. Либо мы можем сделать это непосредственно из Android Studio. Так, нажмем на кнопку Next. И на следующем экране нам будет предложено загрузить Android SDK для последней версии API (в данном случае для Android 11):

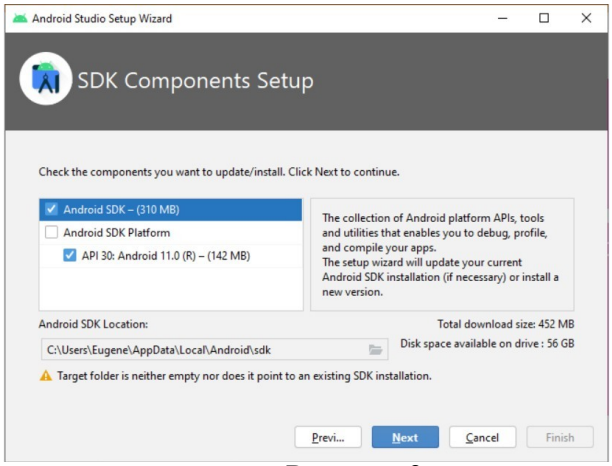


Рисунок 3

Здесь же мы можем указать место для установки Android SDK, если путь по умолчанию нас не устраивает.

Нажмем на кнопку Next, и далее нам отобразится окно со сводкой того, что именно будет установлено:

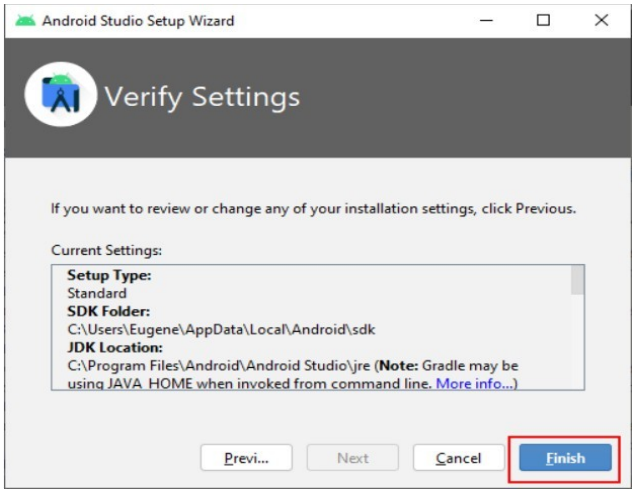


Рисунок 4

Нажмем на кнопку Finish, чтобы, наконец, все это установить.

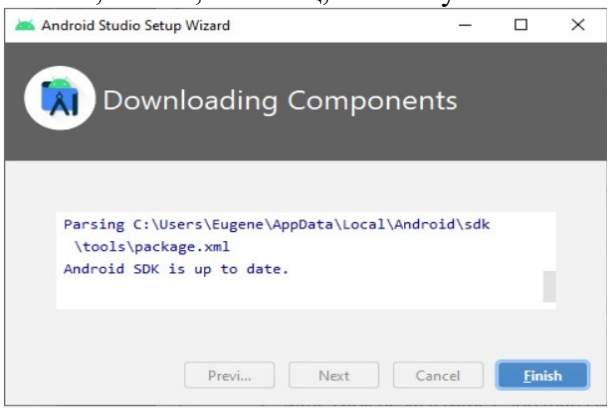


Рисунок 5

И после завершения установки нажмем на кнопку Finish. И мы можем приступать к созданию приложений.

Первый проект в Android Studio

Теперь создадим первое приложение в среде Android Studio для операционной системы Android.

Откроем Android Studio и на начальном экране выберем пункт Create New Project:

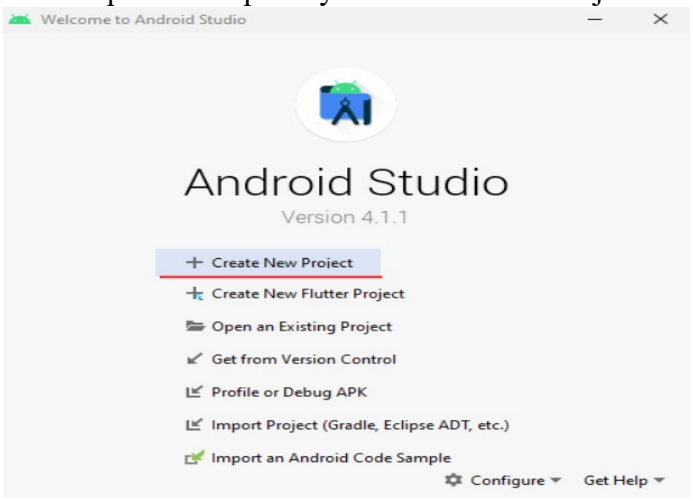


Рисунок 6

При создании проекта Android Studio вначале предложит нам выбрать шаблон проекта:

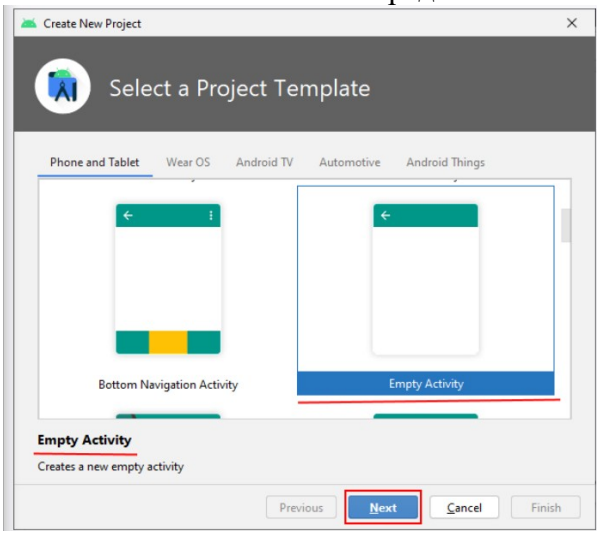


Рисунок 7

Android Studio предоставляет ряд шаблонов для различных ситуаций, но самыми распространенными являются Basic Activity и Empty Activity. Это самые удобные шаблоны для старта для создания большинства приложений. И по умолчанию выбран шаблон Empty Activity (если он не выбран, выберем его) и нажмем на кнопку Next.

После этого отобразится окно настроек нового проекта:

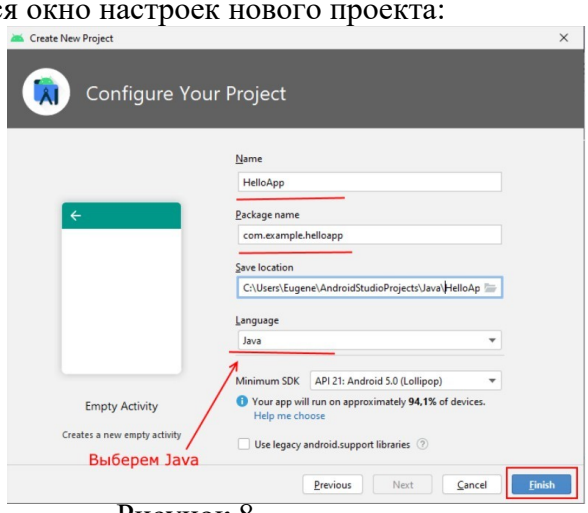


Рисунок 8

В окне создания нового проекта мы можем установить его начальные настройки:

В поле Name вводится название приложения. Укажем в качестве имени название HelloApp

В поле Package Name указывается имя пакета, где будет размещаться главный класс приложения. В данном случае для тестовых проектов это значение не играет ольшого значения, поэтому установим com.example.helloapp.

В поле Save Location установливается расположение файлов проекта на жестком диске. Можно оставить значение по умолчанию.

В поле Language в качестве языка программирования укажем Java (будьт внимательны, так как по умолчанию в этом поле стоит Kotlin)

В поле Minimum SDK указывается самая минимальная поддерживаемая версия SDK. Оставим значение по умолчанию - API 21: Android 5.0 (Lollipop), которая означает, что наше приложение можно будет запустить начиная с Android 5.0, а это 94% устройств. На более старых устройствах запустить будет нельзя.

Стоит учитывать, что чем выше версия SDK, тем меньше диапазон поддерживаемых устройств. Далее нажмем на кнопку Finish, и Android Studio создаст новый проект:

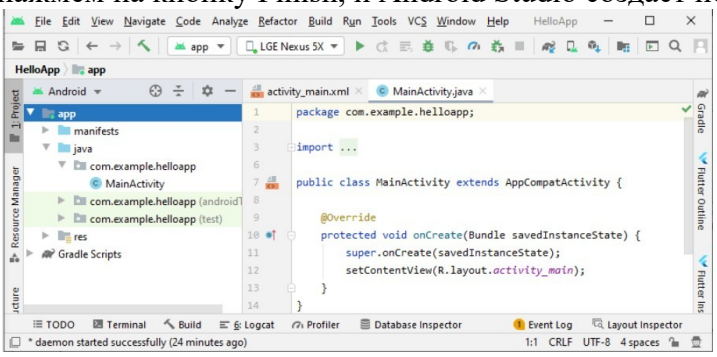


Рисунок 9

Вначале вкратце рассмотрим структуру проекта, что он уже имеет по умолчанию

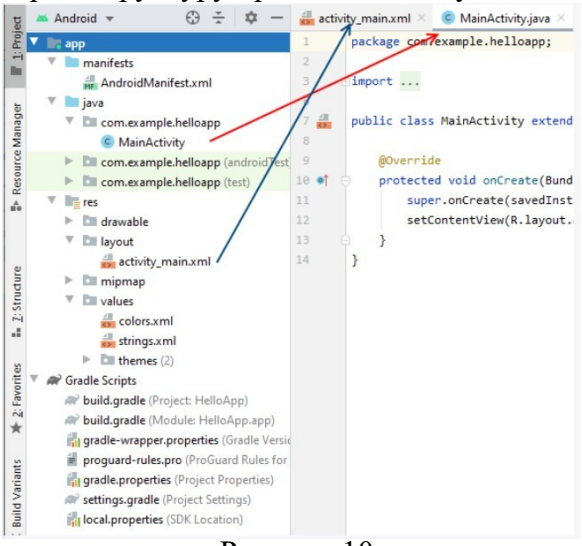


Рисунок 10

Проект Android может состоять из различных модулей. По умолчанию, когда мы создаем проект, создается один модуль - app. Модуль имеет три подпапки:

- manifests: хранит файл манифеста AndroidManifest.xml, который описывает конфигурацию приложения и определяет каждый из компонентов данного приложения.
- java: хранит файлы кода на языке java, которые структурированы по отдельным пакетам. Так, в папке com.example.helloapp (название которого было указано на этапе создания проекта) имеется по умолчанию файл MainActivity.java с кодом на языке Java, который представляет класс MainActivity, запускаемый по умолчанию при старте приложения
- res: содержит используемые в приложении ресурсы. Все ресурсы разбиты на подпапки.
- папка drawable предназначена для хранения изображений, используемых в приложении

– папка layout предназначена для хранения файлов, определяющих графический интерфейс. По умолчанию здесь есть файл activity_main.xml, который определяет интерфейс для класса MainActivity в виде xml
– папки mipmap содержат файлы изображений, которые предназначены для создания иконки приложения при различных разрешениях экрана.
– папка values хранит различные xml-файлы, содержащие коллекции ресурсов - различных данных, которые применяются в приложении.

По умолчанию здесь есть два файла и одна папка:

– файл colors.xml хранит описание цветов, используемых в приложении
– файл strings.xml содержит строковые ресурсы, используемые в приложении
– папки themes хранит две темы приложения - для светлую (дневную) и темную (ночную)

Отдельный элемент Gradle Scripts содержит ряд скриптов, которые используются при построении приложения.

Во всей этой структуре следует выделить файл MainActivity.java, который открыт в Android Studio и который содержит логику приложения и собственно с него начинается выполнение приложения. И также выделим файл activity_main.xml, который определяет графический интерфейс - по сути то, что увидит пользователь на своем смартфоне после загрузки приложения.

Запуск проекта

Созданный выше проект уже содержит некоторый примитивный функционал. Правда, этот функционал почти ничего не делает, только выводит на экран строку "Hello world!". Тем не менее это уже фактически приложение, которое мы можем запустить.

Для запуска и тестирования приложения мы можем использовать эмуляторы или реальные устройства. Но в идеале лучше тестировать на реальных устройствах. К тому же эмуляторы требуют больших аппаратных ресурсов, и не каждый компьютер может потянуть требования эмуляторов. А для использования мобильного устройства для тестирования может потребоваться разве что установить необходимый драйвер.

Режим разработчика на телефоне

По умолчанию опции разработчика на смартфонах скрыты. Чтобы сделать их доступными, надо зайти в Settings > About phone (Настройки > О телефоне) (в Android 8 это в Settings > System > About phone (Настройки > Система > О телефоне)) и семь раз нажать Build Number (Номер сборки).

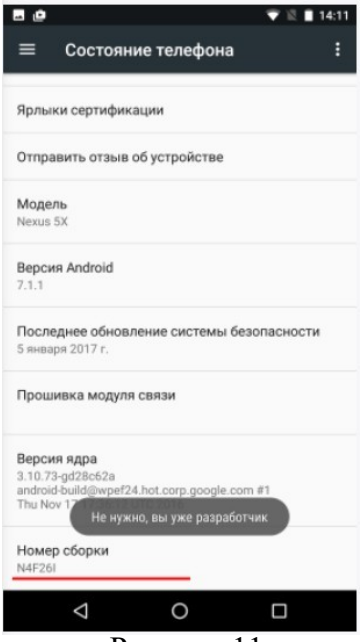


Рисунок 11

Вернитесь к предыдущему экрану и там вы увидите доступный пункт Developer options (Для разработчика).

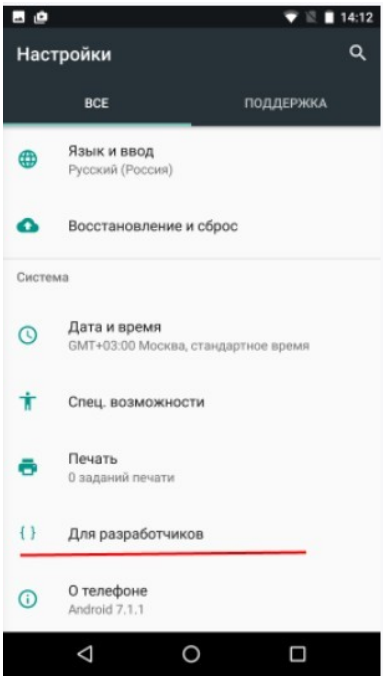


Рисунок 12

Перейдем к пункту Для разработчиков и включим возможность отладки по USB:

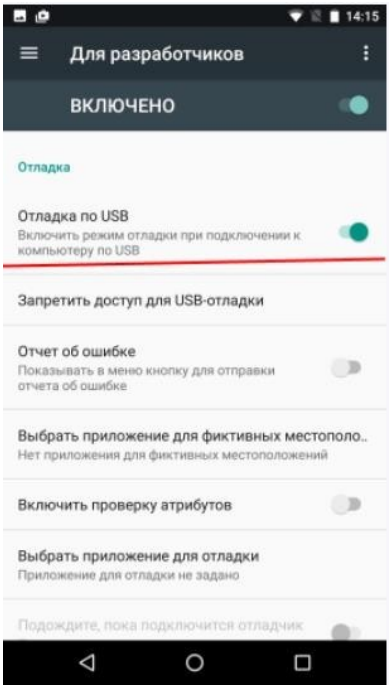


Рисунок 13

Запуск приложения

Подключим устройство с ОС Android (если мы тестируем на реальном устройстве) и запустим проект, нажав на зеленую стрелочку на панели инструментов.

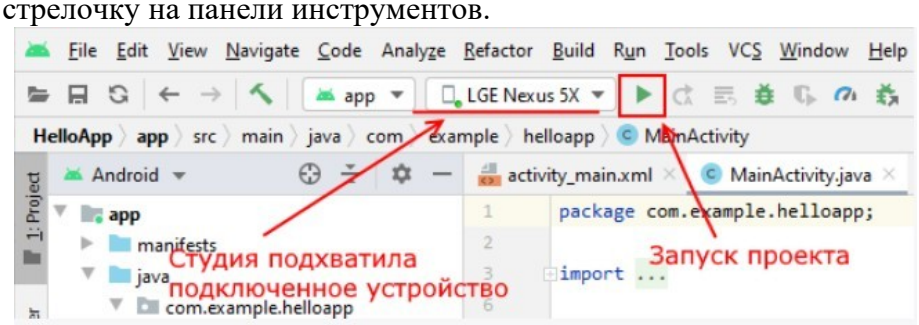


Рисунок 14

Выберем устройство и нажмем на кнопку OK. И после запуска мы увидим наше приложение на экране устройства:

Рисунок 15

Практическая работа № 2.

Тема: Java 2 MicroEdition

Цель работы: получения практических навыков работы с инструментальными средствами J2ME. Установка платформы J2SE

Для установки Sun ONE Studio IDE требуется наличие в системе платформы Java SE версии не ниже 1.4.0. Для проверки версии в окне команд выполните

C:\>java -version

Результат должен выглядеть примерно так

C:\>java -version java version "1.4.0" Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0-b92) Java HotSpot(TM) Client VM (build 1.4.0-b92, mixed mode)

Для установки платформы J2SE v. 1.4.0 в ОС Microsoft Windows, выполните следующие шаги:

1. Поместите файл j2sdk-l_4_0-win.exe (его можно скачать с сайта http://www.sun.com) в каталог, отведенный вами для этой цели, например C:\Java2.
2. Деинсталлируйте предыдущую версию J2SE. если она присутствует в системе.
3. Запустите инсталлятор j2sdk-l_4_0-win.exe.
4. Следуйте инструкциям на экране до завершения процесса

установки. Установка IDE

1. Поместите файл ffj_me_win32.exe (его можно получить С http: //www. sun. com/) в каталог, отведенный для Sun ONE Studio IDE, например. C:\Java2\studio.
2. Запустите инсталлятор f f j_me_win32 . exe.
3. Следуйте инструкциям.
4. Укажите каталог с ПО Java 2 SDK.
5. Укажите каталог для инсталляции. Имя каталога не до.лжно содержать пробелов и каталог должен быть пустым или новым.
6. Нажмите Next для подтверждения списка модулей. Должны быть установлены Core Platform and Modules и J2ME Wireless Toolkit
7. Подтвердите настройки. Начнется процесс инсталляции.
8. Можно установить связь расширений . j ava и . nbm с Sun ONE Studio IDE.
9. Завершите инстал-

ляцию. Настройка IDE

1. Запустите Sun ONE Studio IDE.

2. При первом запуске программа запросит ваш пользовательский каталог. Введите имя каталога, в котором хотите хранить свои настройки, информацию о проектах и нажмите ОК. Этот каталог должен отличаться от каталога, в который установлено IDE.
3. Выберите режим окна.
4. Выберите активные модули (по умолчанию - все).
5. Выберите режим автоматического обновления.
6. Выберите метод регистрации.

Проверка установленного Sun ONE Studio IDE

1. Установленный Sun ONE Studio IDE можно проверить на одном из тестовых приложений.
2. Откройте каталог примеров (ffj-user-dir/sampledir) в окне Explorer IDE.
3. Кликните комплект мидлетов правой клавишей мыши и выберите Execute.
4. Если Sun ONE Studio IDE установлен правильно, запустится эмулятор мобильного устройства, установленный по умолчанию, с выбранным мидлетом на экране.

Монтирование файловой системы

1. Начните работу с монтирования файловой системы или каталога, в котором предполагается размешать код приложений. Монтирование позволит вам просматривать файлы и работать с ними в рамках Sun ONE Studio IDE. Монтированная файловая система включается в путь к классам Java, что необходимо для компиляции, запуска и отладки.
2. Чтобы монтировать файловую систему из меню File выберите Mount Filesystem. Это откроет окно выбора шаблона файловой системы. Выберите Local Directory и нажмите Next. Укажите требуемый каталог и нажмите Finish.
3. Выбранный каталог появится на вкладке Filesystems панели Explorer. Каталог смонтирован и готов к работе.

Установка эмулятора по умолчанию

Эмулятор позволяет симулировать исполнение приложения на целевом устройстве. Обычно эмулятор включает примеры устройств, которые он эмулирует. Такие образцы называются “skins”. Эмулятор J2ME Wireless Toolkit Emulator входит в поставку Sun ONE Studio и содержит несколько примеров устройств, например, телефон Motorola i85.

Можно создать несколько экземпляров эмулятора, что позволит быстрее и проще отлаживать приложение для различных устройств.

Эмулятор по умолчанию - один из установленных эмуляторов, который используется для запуска приложений, которые не являются частью комплектов мидлетов и для новых комплектов мидлетов. Для установки эмулятора по умолчанию:

1. На вкладке Runtime панели Explorer раскройте узел Installed Emulators и кликните правой кнопкой мыши на эмуляторе. Выберите Set As Default.
2. В панели Explorer кликните правой кнопкой мыши на первом экземпляре J2ME Wireless Toolkit и выберите Properties из меню.
3. Установите свойства для этого эмулятора.

Для установки одного из примеров устройств в качестве устройства по умолчанию для эмулятора:

1. На вкладке Runtime панели Explorer разверните Installed Emulators, затем разверните J2ME Wireless Toolkit. Вы увидите список устройств (skins).
2. Кликните правой клавишей мыши на одном из устройств и выберите Set as

Default. Создание пакетов для мидлета

Для создания пакета:

1. В панели Explorer на вкладке Filesystems кликните правой клавишей мыши каталог и выберите New \ Java Package. Новый пакет можно разместить в любом каталоге файловой системы.
2. В окне New Wizard введите имя нового пакета и нажмите Finish. Созданный пакет будет отображаться в панели Explorer с иконкой в виде папки.

Создание мидлета

1. В панели Explorer кликните правой клавишей мыши на имени пакета. В контекстном меню выберите New \ MIDP \ MIDlet.
2. Выбор шаблона мидлета запустит wizard для создания мидлета.
3. Введите имя мидлета. нажмите Next.
4. Нажмите Finish для создания мидлета.
5. Созданный мидлет появится на вкладке Filesystem панели Explorer, его код появится в окне Source Editor. Иконка мидлета представляет сетку нулей и единиц, это показывает, что мидлет еще не скомпилирован.
6. Разверните узел мидлета для просмотра его классов, а также их полей, конструкторов и методов.

Программирование мидлета

Вы можете писать код мидлета двумя способами: непосредственно вводя исходный код в окне Source Editor пли используя инструменты для создания методов, полей, конструкторов, инициализаторов, классов и интерфейсов.

Например, добавление нового поля к классу можно выполнить так:

1. На вкладке Filesystem панели Explorer кликните правой клавишей мыши на имени класса и выберите Add \ Field.
2. Заполните диалоговое окно Add New Field.
   a. Введите имя нового поля, выберите его тип.
   b. Установите модификаторы поля и спецификаторы доступа.
   c. Установите исходное значение поля.
   d. Нажмите ОК.
3. Двойной клик на имени класса откроет его код в окне Source Editor.
4. Введите необходимые описания и комментарии. Компиляция мидлета

Процесс компиляции включает в себя собственно компиляцию исходного кода и создание бинарных файлов .class, а также предварительную проверку созданных классов и оптимизацию кода для виртуальной машины CLDC.

Для компиляции мидлета:

1. В панели Explorer кликните правой клавишей мыши на имени мидлета и выберите Compile. Также можно использовать пункты меню Compile или Build для компиляции класса мидлета и всех связанных с ним классов.
2. Просмотрите сообщения компилятора в окне Output.
3. Если в процессе компиляции произошли ошибки, класс в панели Explorer будет помечен иконкой с буквой “X”. В окне Output отображается строка(и) кода, содержащая ошибку и эта строка подсвечивается в окне Source Editor.

Тестирование мидлета

Для тестирования мидлета после успешной компиляции используйте пункт меню Execute, который запустит мидлет на исполнение с использованием эмулятора по умолчанию.

Создание комплекта мидлетов (MIDIet Suite)

Комплект мидлетов организует исходные файлы и файлы атрибутов приложения MIDP. При создании комплекта мидлетов автоматически создается необходимый файл JAR. который содержит файлы приложения. Также Sun ONE Studio IDE создает файл описателя приложения JAD.

Для создания комплекта мидлетов:

1. Создайте новый каталог, в котором будет размешаться комплект мидлетов.
2. В панели Explorer в контекстном меню созданного каталога выберите New \ MIDP \ MIDIet Suite.
3. В окне MIDletSuite wizard введите имя нового комплекта мидлетов. нажмите Next.
4. Выберите опцию Use Existing MIDIet и введите имя класса мидлета для включения его в комплект.
5. Установите свойства мидлета.
6. Выберите эмулятор по умолчанию для мидлета. нажмите Finish.
7. Разверните узел комплекта и проверьте его содержимое. Наполнение комплекта мидлетов

1. В контекстном меню комплекта выберите Edit Suite, появится соответствующее окно. Выберите вкладку Jar Contents для просмотра содержимого файла JAR.
2. Выберите классы, которые надо добавить к комплекту и нажмите Add. Используйте кнопки Remove и Remove АП для удаления содержимого из JAR.
3. Используйте пункт Properties контекстного меню комплекта для настройки его свойств.
4. Выберите пункт Update JAR контекстного меню комплекта для активизации внесенных изменений.
5. Можно просмотреть содержимое файла JAD и декларации JAR с помощью пункта View Manifest/JAD контекстного меню комплекта.

Тестирование комплекта мидлетов

1. В контекстном меню комплекта мидлетов выберите пункт Compile или Compile АП (или Build или Build АП) для компиляции приложений в комплекте.
2. В контекстном меню комплекта мидлетов выберите пункт Execute для запуска комплекта.

Обратите внимание, что приложение запускается на эмуляторе и устройстве, выбранном по умолчанию для данного комплекта.

Вместо запуска конкретного приложения на эмуляторе будет отображен список приложений комплекта для выбора пользователем.

Задания

1. Установите среду разработки Sun ONE Studio IDE.
2. Проведите сборку одного из комплектов мидлетов в каталоге примеров, запустите его на выполнение на эмуляторе различных устройств.
3. Создайте простой мидлет. отображающий список из нескольких элементов и имеющий команд}' завершения работы.
4. Сформируйте комплект мидлетов из 1-2 мидлетов, обратите внимание на различное поведение эмулятора при обработке комплекта и отдельных мидлетов.

Контрольные вопросы по теории

1. Дайте определение пакета, мидлета и комплекта мидлетов.
2. Опишите их назначение
3. Опишите их взаимосвязь.

Практическая работа № 3.3. Протокол Bluetooth

Цель работы: научиться применять устройства Bluetooth для передачи информации: физически устанавливать соединение и программно реализовывать передачу данных.

Bluetooth – это технология передачи данных по радиоканалам на короткие расстояния, позволяющая осуществлять связь беспроводных телефонов, компьютеров и различной периферии даже в тех случаях, когда нарушается требование прямой видимости.

Первый стандарт Bluetooth 1.0. был выпущен в декабре 1999 г. Технология названа в честь датского короля Гарольда Блатана (пер. с датского «синий зуб» на английский «blue tooth»), объединившего в X веке земли Дании и Норвегии и прославившийся тем, что умел находить общий язык с князьями-вассалами.

Первоначально технология задумывалась как средство простого соединения ПК и телекоммуникационных устройств мобильных телефонов. Но оказалась настолько удачной, что ее развитие видится весьма перспективным.

Специалисты предполагают два направления использования Bluetooth. Первое — домашние сети, включающие в себя различную электронную технику, в частности компьютеры, телевизоры и т. п. Второе — локальные сети офисов небольших фирм, где стандарт Bluetooth может прийти на смену традиционным проводным технологиям.

Расстояние, на которое может быть установлено соединение Bluetooth – невелико и составляет от 10 до 30 м. В настоящее время разработчики пытаются его увеличить хотя бы до 100 м. Зато для Bluetooth не требуется прямой видимости или направленной антенны, соединение может быть установлено через стену, при условии, что она не экранирована.

Главной особенностью интерфейса Bluetooth является то, что поддерживающие его устройства соединяются друг с другом автоматически, стоит им только оказаться в пределах досягаемости.

Основу архитектуры Bluetooth составляет пикосеть (piconet), состоящая из одного главного узла (M) и нескольких (до семи) подчиненных узлов (S), расположенных в радиусе 10 м. В одной и той же комнате, если она достаточно большая, могут располагаться несколько пикосетей.

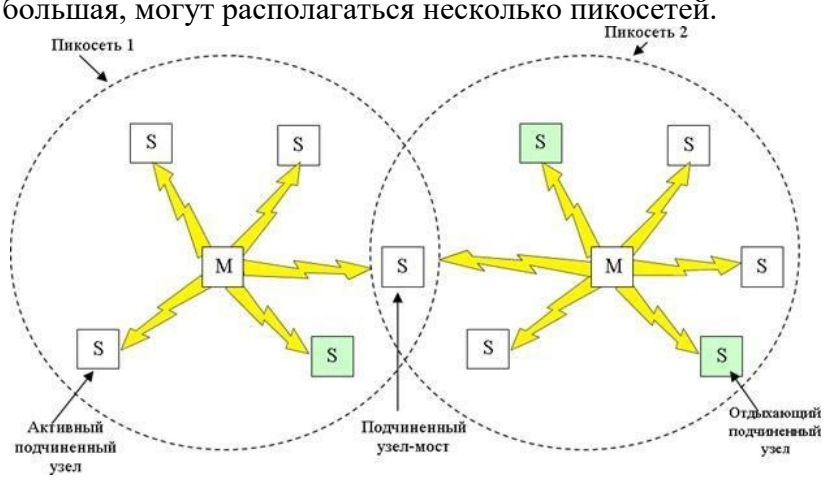


Рисунок 16

Несколько объединенных вместе пикосетей составляют рассеянную сеть (scatternet). Помимо семи активных подчиненных узлов, один главный узел может поддерживать до 255 так называемых отдыхающих узлов. Это устройства, которые главный узел перевел в режим пониженного энергопотребления (за счет чего продлевается ресурс их источников питания).

В таком режиме узел может только отвечать на запросы активации или на сигнальные последовательности от главного узла. Существуют еще два промежуточных режима энергопотребления – приостановленный и анализирующий.

Такое решение с главным и подчиненным узлами оказалось очень простым и дешевым в реализации, несмотря на то, что подчиненные узлы получились неразговорчивыми — они лишь выполняют то, что им прикажет главный узел.

В основе пикосетей лежит принцип централизованной системы с временным уплотнением. Главный узел контролирует временные интервалы и распределяет очередность передачи данных каждым из подчиненных узлов. Связь существует только между подчиненным и главным узлами. Прямой связи между подчиненными узлами нет.

Большинство сетевых протоколов просто предоставляют каналы связи между коммуникационными единицами и оставляют прикладное использование этих каналов на усмотрение разработчиков. В противоположность этому, например, спецификация Bluetooth V1.1, называет 13 поддерживаемых приложений и для каждого из них предоставляет свой набор протоколов, называемые профилями.

Профиль – основа, на которой строятся реальные приложения. Его главная задача состоит в создании канала между главным и подчиненным узлами.

Довольно общим является профиль определения сервиса, используемый для определения того, какие сервисы могут быть предоставлены другими устройствами. Вся аппаратура системы Bluetooth должна поддерживать два описанных ниже профиля. Все прочие являются необязательными.

Профиль последовательного порта — это транспортный протокол, который используется большинством других профилей. Он эмулирует последовательный канал и полезен при работе с приложениями, которым требуется этот канал.

Профиль общего объектного обмена определяет клиент-серверные взаимоотношения, возникающие при обмене данными. Клиенты инициируют операции, но подчиненная станция может выступать либо в роли клиента, либо в роли сервера.

Следующая группа профилей имеет отношение к сетям. Профиль доступа к ЛВС позволяет устройству Bluetooth подсоединиться к стационарной вычислительной сети. Он позволяет ноутбуку соединяться с мобильным телефоном, имеющим встроенный модем.

Профиль «Факс» позволяет беспроводным факс-машинам отсылать/получать факсы при помощи мобильного телефона.

Три профиля относятся к телефонии. Профиль беспроводной телефонии обеспечивает связь телефонной трубки с базой. Сейчас домашний телефон не может использоваться в качестве мобильного, даже если он не имеет совсем никаких проводов, однако в будущем, эти два устройства обязательно будут объединены.

Профиль Intercom позволяет двум телефонам соединяться друг с другом наподобие раций. Профиль Гарнитура представляет собой приложение, позволяющее устройствам hands-free держать связь с базой (телефоном).

Последние три профиля (Передача объектов, Передача файлов, Синхронизация) предназначены для организации обмена данными между беспроводными устройствами. Объекты могут представлять собой электронные визитные карточки, изображения или файлы с данными.

Пока недостатком технологии Bluetooth является сравнительно низкая скорость передачи данных (720 кбит/с), поэтому она не способна обеспечить передачу видеосигнала.

Начиная с 2001 г. вышло несколько версий Bluetooth. Наиболее существенным изменением в технологии стало введение аутентификации и, как следствие, шифрования передаваемых файлов с целью усиления системы защиты беспроводной сети.

Bluetooth-устройство включает в себя радиоприемник и радиопередатчик, которые работают в диапазоне частот от 2400 до 2483,5 МГц (этот диапазон в большинстве стран мира является открытым и свободным от лицензирования). Используемые частоты определяют возможности Bluetooth по передаче данных. Ширина канала для устройств Bluetooth составляет 723,2 кбит/с в асинхронном режиме.

Прежде чем установить сеанс связи, устройства Bluetooth должны обменяться криптографическим ключами, подтверждающими идентичность устройства и его владельца. Дальнейшие операции разрешаются только при совпадении ключей обеих сторон. При этом используется упрощенный механизм аутентификации, предполагающий, что каждое устройство выполняет в этом процессе роль либо ведущего (master), генерирующего ключ, либо ведомого (slave), осуществляющего проверку ключа.

Задание 1. Подключите и настройте устройства Bluetooth

1. Физически подключите устройства Bluetooth к USB-разъемам компьютеров. Если ранее это устройство не устанавливалось на компьютере, то будет произведен поиск драйверов и их установка. В итоге в области уведомления появится значок .
2. Откройте диалоговое окно Устройства Bluetooth (Пуск/Панель управления/Устройства Bluetooth).
3. Установите Bluetooth-имя для первого компьютера:
   перейдите на вкладку Оборудование;
   дважды щелкните по элементу USB Bluetooth Wireless Adapter;
   на вкладке Дополнительно, введите имя этого компьютера (Comp_1);
   примените параметры кнопкой OK.
4. Аналогично установите Bluetooth-имя Comp_2 для второго компьютера.
5. Разрешите обнаружение компьютеров другими устройствами Bluetooth, установив на вкладке Параметры флажок Включить обнаружение.
6. Выполните обнаружение Bluetooth-устройств:

   перейдите на вкладку Устройства;
   щелкните по кнопке Добавить;
   установите флажок Устройство установлено и готово к обнаружению и закройте окнокнопкой Далее.
   После этого появится список обнаруженных устройств. Там должен отображаться другой компьютер.
   выберите другой компьютер (Comp_1) и щелкните по кнопке Далее;
   выберите Выбрать ключ доступа самостоятельно и введите в поле любое значение (например, 123). Закройте окно кнопкой Далее. На другом компьютере будет выведено сообщение о необходимости ввести ключ доступа.
   Введите тот же ключ, что и на первом компьютере - (123) и закройте текущее окно кнопкой Далее.
   сбросьте флажок Выключить обнаружение;
   примените параметры кнопкой Готово. Задание 2. Передайте файл с одного компьютера на другой:

1. Подготовьте на первом компьютере файл в формате RTF, содержащий сведения о технологии

Bluetooth.

2. Подготовьте второй компьютер для приема файлов. Для этого щелкните по значку и выберите команду Принять файл.
3. Передайте файл:

– вызовите диалоговое окно Мастера передачи файлов на первом компьютере (щелкните по значку и выберите Отправить файл);
– выберите получателя вашего файла, для чего щелкните по кнопке Обзор и выберите имя другого компьютера, щелкните Далее;
– выберите файл для передачи, для этого щелкните по кнопке Обзор и перейдите в папку с файлом и дважды щелкните по нужному файлу;
– инициализируйте передачу кнопкой Далее;
– завершите передачу файла кнопкой Готово.

4. Примите полученный файл на другом компьютере:
   – щелкните по кнопке Обзор и перейдите в вашу папку, щелкните Далее;
   – завершите приме файла кнопкой Готово.
5. Проверьте полученный файл, открыв его в текстовом процессоре.

Задание 3. Установите и настройте Личную сеть PAN):

1. Откройте окно свойств Bluetooth:

– на первом компьютере откройте окно Сетевые подключения;
– вызовите контекстное меню элемента Сетевое подключение Bluetooth и выберите команду Свойства.

2. Перейдите на вкладку Общие.
3. Установите адрес этого компьютера вручную:

– откройте диалоговое окно свойств элемента Протокол Интернета (TCP/IP);
– выберите Использовать следующий IP-адрес;
– введите в поле IP-адрес – 192.168.0.1;
– введите в поле Маска подсети – 255.255.255.0;
– примените параметры кнопкой OK.

4. Закройте текущее окно кнопкой OK.
5. Аналогично установите адрес другого компьютера. Используйте следующие данные:
   - IP-адрес – 192.168.0.2;
   - Маска подсети – 255.255.255.0.
6. Подключите Личную сеть Bluetooth:

– вызовите контекстное меню индикатора Bluetooth в области уведомления и выберите команду Присоединиться к личной сети (PAN);
– выберите другой компьютер и щелкните Подключить. Задание 4.

Выполните самостоятельные задания:

1. Определите время, необходимое для передачи файла размером 3 Мб, и оформите результат в виде таблицы:

Таблица 2

Способ передачи	Объем файла (Мбайт)	Время затраченное на передачу (в секундах)	Теоретическая скоростьпередачи (бит/с)	Расчетная скорость передачи (бит/с)
Инфракрасная связь				
Bluetooth				

2. Настройте личную сеть Bluetooth для работы по динамически формируемым адресам.
3. Организуйте обмен данными между двумя мобильными телефонами посредством Bluetooth.
4. Организуйте передачу информации с мобильного телефона на ПК и обратно.

Вопросы для проверкти

1. В чем суть технологии Bluetooth?
2. Опишите направления использования Bluetooth .
3. В чем состоит главная особенность интерфейса Bluetooth?
4. Опишите пикосеть и рассеянную сеть.
5. Что такое профиль? Опишите основные профили.

Практическая работа № 4. Установка инструментария для разработки мобильных приложений

Практическая работа № 5. Настройка среды для разработки мобильных приложений

Цель работы: выполнить установку среды разработки и настроить ее для работы

Установка и настройка компонентов среды разработки

Приложения для Android, как и большинство приложений для коммуникаторов, разрабатываются на стандартном ПК, где ресурсы не ограничены (по сравнению с мобильным устройством) и загружаются на целевой коммуникатор для отладки, тестирования и последующего использования. Приложения можно отлаживать и тестировать на реальном устройстве под управлением Android или на эмуляторе. Для первоначальной разработки и отладки удобнее использовать эмулятор, а затем выполнять окончательное тестирование на реальных устройствах.

Разработчика предоставляется возможность использовать средства разработки приложений для Android на ПК под управлением любой из распространенных операционных систем: Windows, Linux и Mac OS X. Ниже будет детально описан процесс установки компонентов среды разработки под Windows XP, для прочих операционных систем отличия весьма незначительны.

Установка JDK

Для Android SDK требуется JDK версии не ниже 5 (на момент написания данного методического руководства была актуальна 7-я версия Sun JDK и 6-я версия Open-JDK). Для Windows традиционно используется Sun JDK, который можно бесплатно загрузить на сайте разработчика:

Страница загрузки Sun JDK http://www.oracle.com/technetwork/java/javase/downloads/index.html

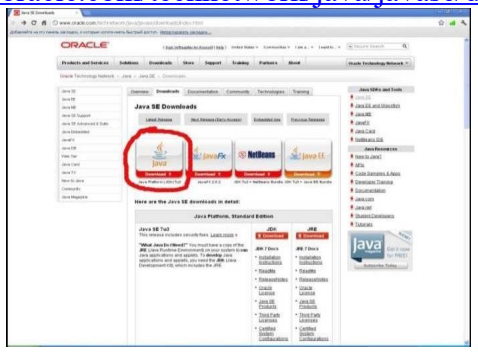


Рисунок 17

Рисунок 18

Принятия условий лицензионного соглашения Oracle Binary Code License Agreement for Java SE

(1) можно выбрать подходящую для вашей платформы ссылку(2) Установка загруженного JDK достаточно проста:

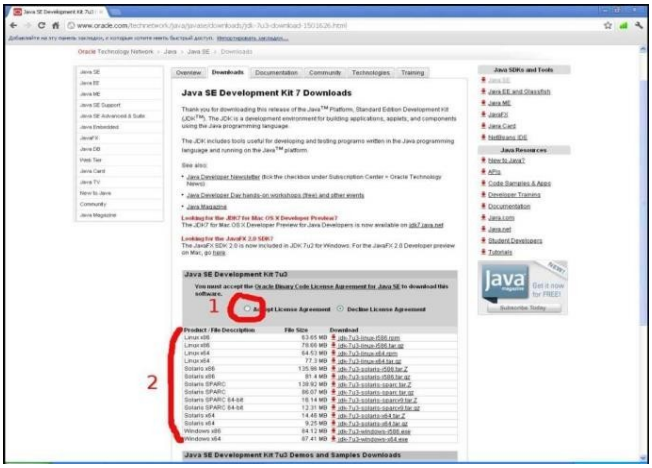


Рисунок 19

каталог для установки по умолчанию (1) не подходит, его можно изменить (2), а затем продолжить установку:

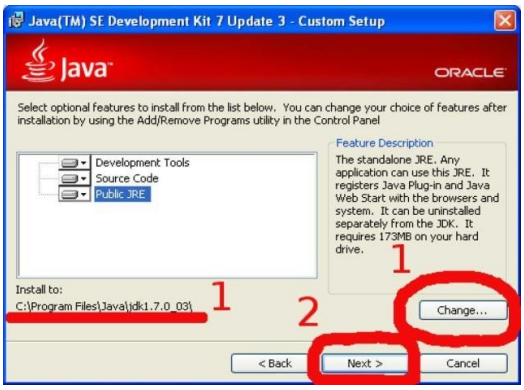


Рисунок 20

При установке Java Runtime так же можно изменить каталог для установки:

Рисунок 21

и продолжить установку:

Рисунок 22

Далее при желании можно зарегистрировать установленный продукт:

Рисунок 23

И установить JavaFX SDK, если не жалко 50 МБ дискового пространства:

Рисунок 24

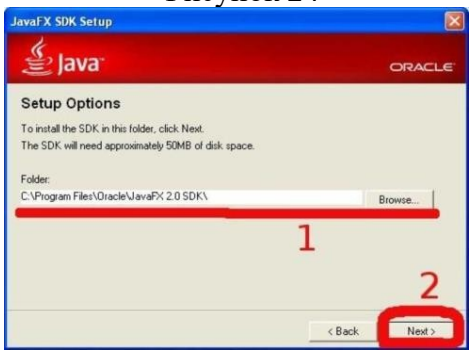


Рисунок 25

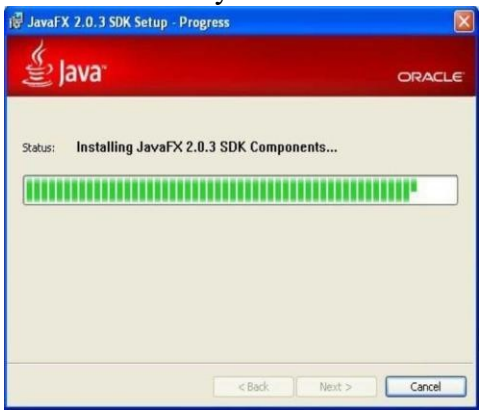


Рисунок 26

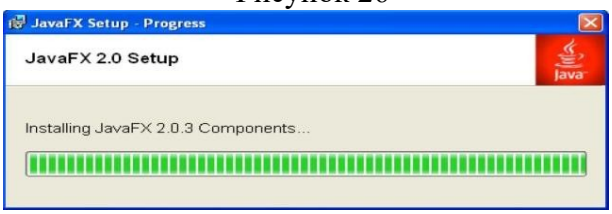


Рисунок 27

И, наконец, установка JDK закончена:

Рисунок 28

Установка Android SDK

На странице http://developer.android.com/sdk выбираем installer_r16-windows.exe

Здесь же имеются дополнительные инструкции по установке, а также ссылка на пошаговую инструкцию по установке SDK на все поддерживаемые платформы:

http://developer.android.com/sdk/installing.html

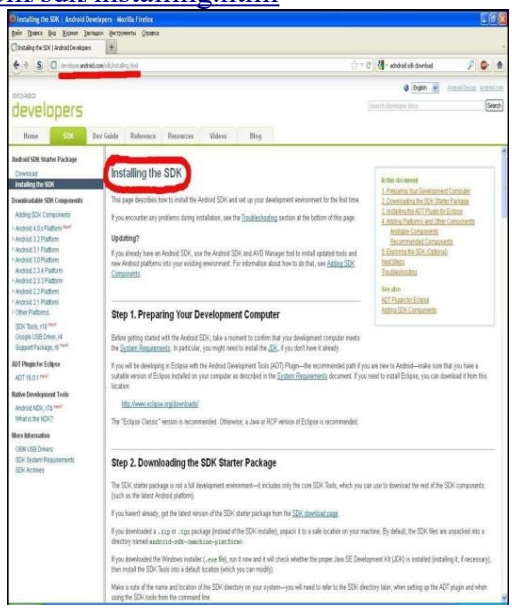


Рисунок 29

Ресурс http://developer.android.com является основным источником информации о платформе Android «из первых рук». В нем содержится огромное количество самых разнообразных сведений, необходимых разработчику, начиная с описания API и кончая такими вещами, как рекомендации по дизайну приложений и повышению производительности приложений. В данном методическом пособии в дальнейшем будут встречаться ссылки на конкретные страницы, посвященные изучаемым темам.

Установка загруженного Android SDK также не отличается особой сложностью:

Рисунок 30

Мастер установки обнаружит (если сможет) установленную версию JDK (1), после чего установку можно продолжить (2):

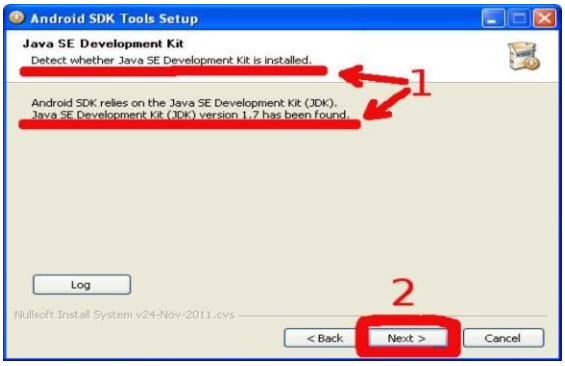


Рисунок 31

Рисунок 32

Установка начинается

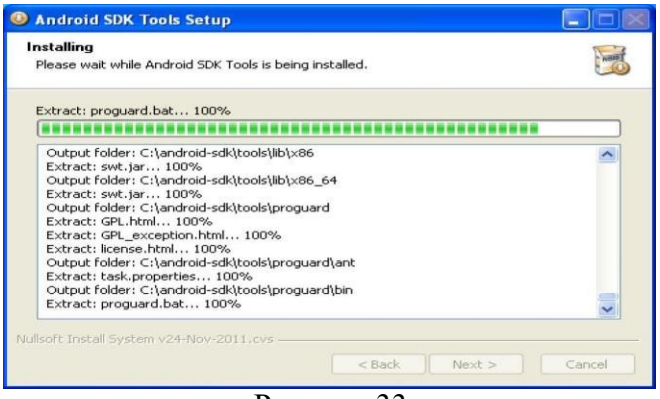


Рисунок 33

и заканчивается:

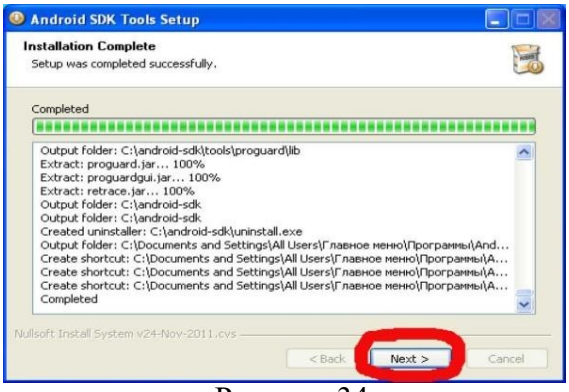


Рисунок 34

Можно сразу поставить галку(1) и запустить SDK Manager для окончательной настройки(2):

Рисунок 35

Требуется загрузить нужные версии API и другие полезные инструменты:

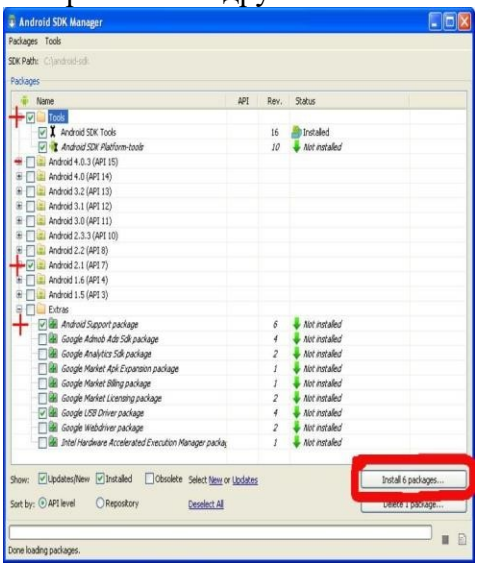
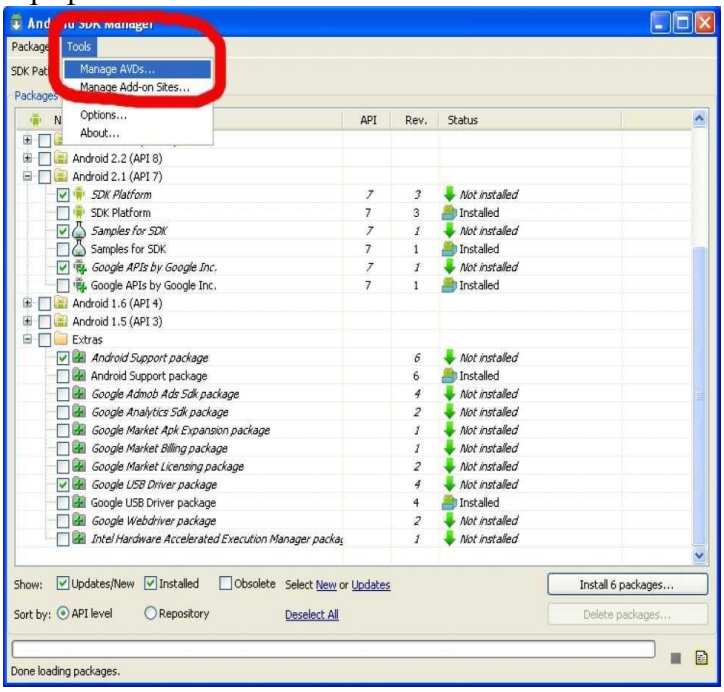


Рисунок 36

Далее мы запустим менеджер AVD (Android Virtual Devices) и сконфигурируем эмулятор на использование Android версии 2.1 с расширением Google API (GAPI). GAPI нужен, как правило, для приложений, использующих картографические возможности Android.



Рисунок

37 Создадим новое виртуальное устройство:

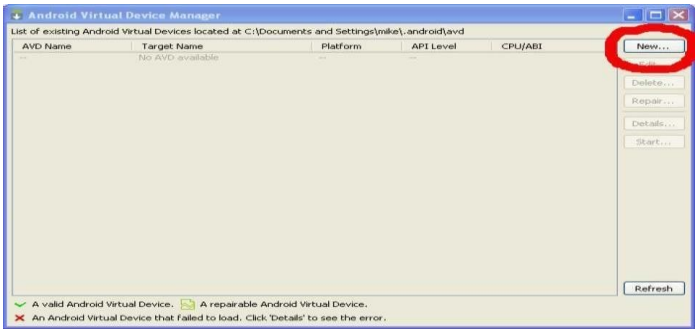


Рисунок 38

При создании нового виртуального устройства выбираем его название(1), целевой API (2), размер (или файл с образом) виртуальной SD-карты (3), одно из стандартных или собственное разрешение экрана (4) и, наконец, создаем устройство(5).

При необходимости можно указать плотность пикселей на экране, максимальный размер кучи для приложений внутри виртуальной машины, а также другие параметры:

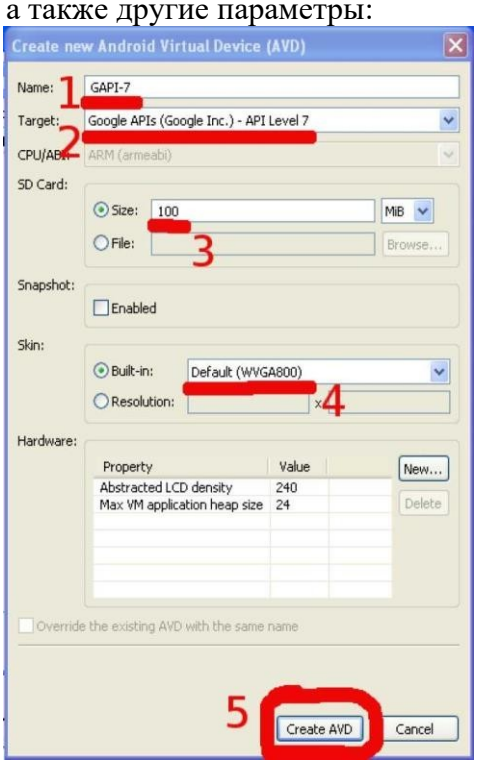


Рисунок 39

Убедимся, что все получилось, как хотели (1) и продолжим работу (2):

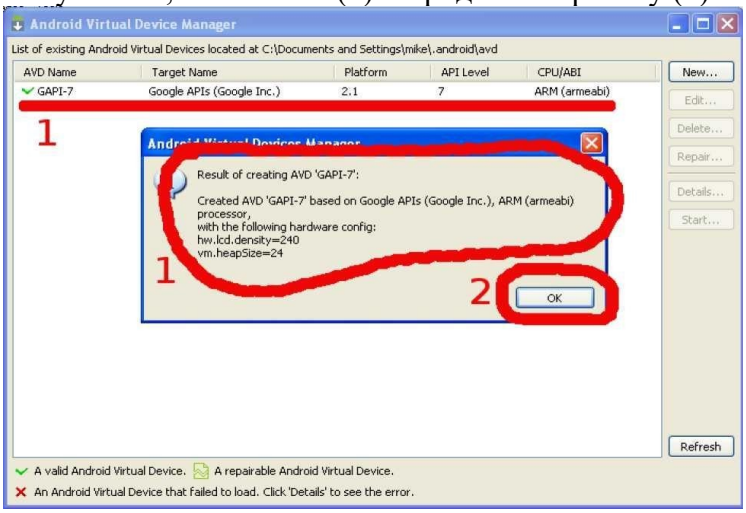


Рисунок 40

Пришло время запустить эмулятор:

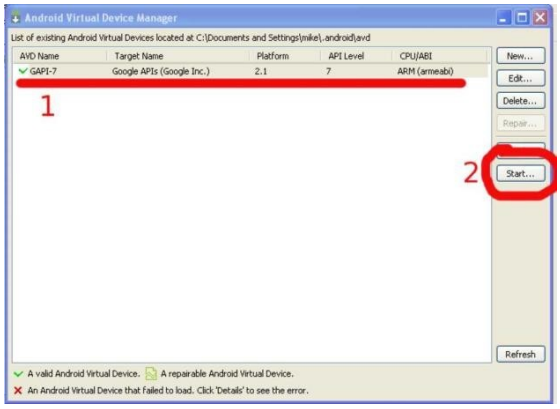


Рисунок 41

Важно: при использовании русскоязычных версий ОС Windows путь к рабочему каталогу AVD может включать русские буквы, что приведет к невозможности запуска эмуляторов. В этом случае можно создать каталог для AVD в корневом каталоге диска (например, C:\AVD) и присвоить переменной окружения ANDROID_SDK_HOME значение этого пути.

При запуске эмулятора из менеджера AVD можно дополнительно указать некоторые параметры, особенно полезно управление размерами или плотностью пикселей экрана виртуального устройства: поставив галку (1), можно указать желаемый размер экрана (2) в дюймах, вычислить, при необходимости, плотность пикселей (3) на используемом мониторе ПК:

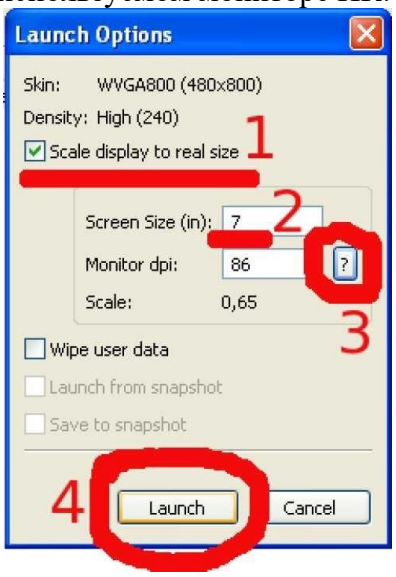


Рисунок 42

При первом запуске вы можете указать, следует ли отправлять анонимную статистику использования SDK в Google:

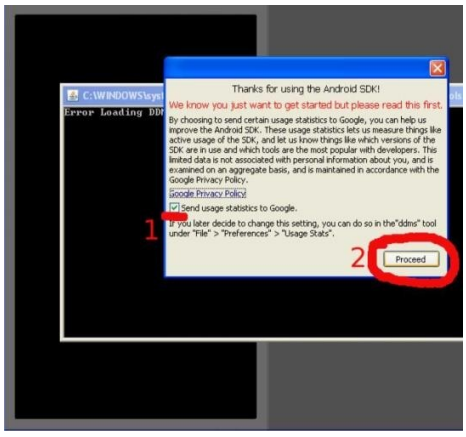


Рисунок 43

Более того, «пощупать» самые новые версии ОС Android и оценить их функциональные возможности можно безвозмездно, то есть даром просто выбрав нужную версию API при создании виртуального устройства.

Одна из неприятных особенностей платформы Android с точки зрения разработчика заключается в большом количестве выпущенных версий, значительно различающихся по возможностям. В настоящий момент Open Handset Alliance (т. е. Google) несколько затормозила этот процесс, и на данный момент флагманской версией является 4.0.x. Тем не менее, при выборе уровня API (API level) для нового ПО следует учитывать наличие у пользователей большого количество относительно старых устройств, так что для охвата максимального сегмента рынка оптимальным выбором является Android 2.1 (поддерживается 97% процентов устройств с Android).

Так же, как и в реальном, в виртуальном устройстве существует опасность разрушить файловую систему при неправильном прекращении работы (сбое питания и т. п.).

Установка IDE Eclipse

Рисунок 44

Интегрированная среда разработки Eclipse (довольно странное название, по-английски означает, в том числе помутнение рассудка) доступна для загрузки на официальном сайте по адресу http://eclipse.org/downloads/ и распространяется в виде ZIP-архива:

В нашем случае подходящим вариантом является Eclipse IDE for Java Developers.

После выбора подходящей для используемой на ПК операционной системы версии IDE, для уменьшения времени загрузки имеет смысл выбрать европейское зеркало сайта:

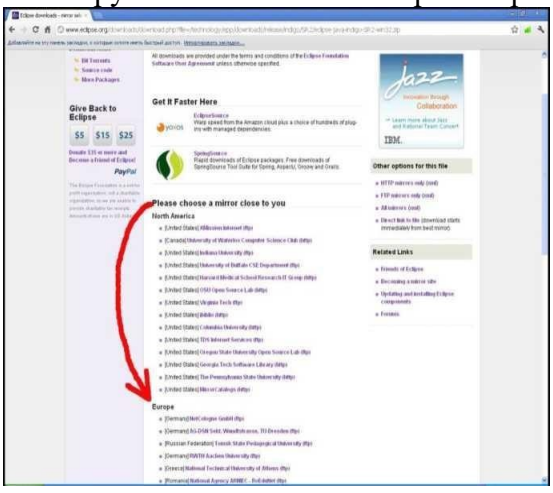


Рисунок 45

Получив нужный архив, выбираем подходящий путь для установки и распаковываем архив:

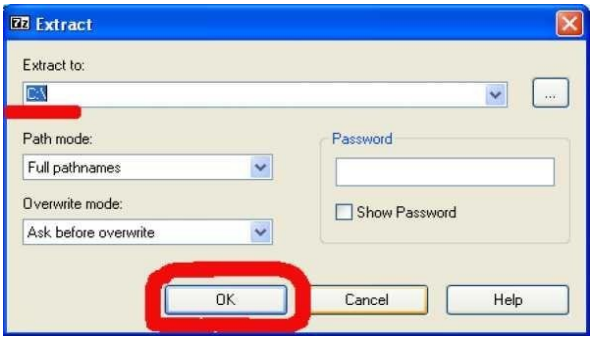


Рисунок 46

Установка плагина ADT

Для установки плагина ADT не требуется ручная загрузка, он будет установлен системой управления плагинами Eclipse. Тем не менее, мы рассмотрим установку подробно.

Запустим Eclipse

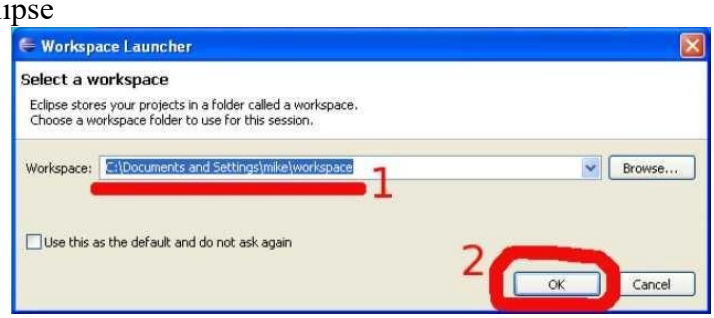


Рисунок 47

Выберем нужный пункт меню

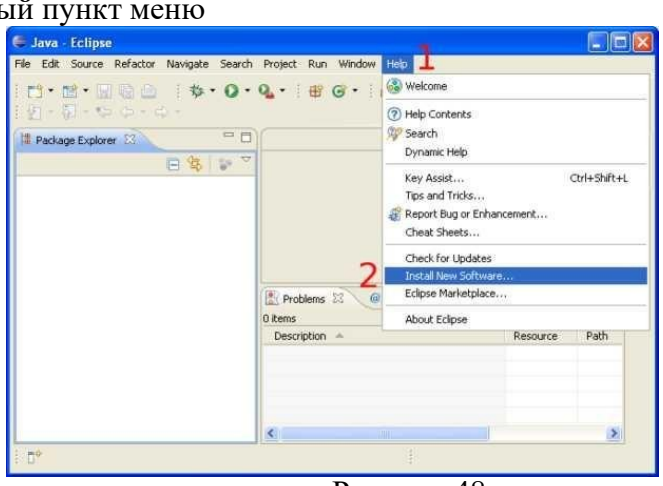


Рисунок 48

После установки выбранных компонентов перезапустим Eclipse

И настроим плагин ADT на использование уже установленного Android SDK.

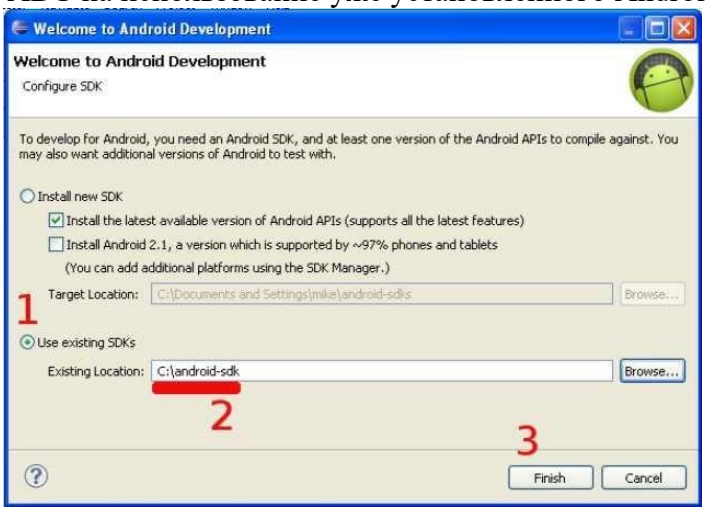


Рисунок 49

На снимке экрана (выше) можно увидеть, что ADT в состоянии сам загрузить и установить Android SDK. Мы использовали ручную установку для ускорения процесса (не тратили время на ожидание загрузки SDK).

Среда разработки установлена, пришло время создать первое приложение для платформы Android и запустить его в эмуляторе.

Создание первого приложения под Android

Для создания приложений в установленной нами среде разработки удобно использовать

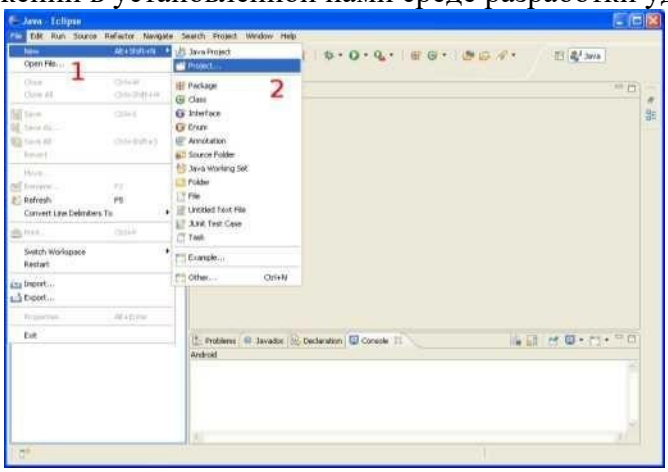


Рисунок 50

«Мастер создания нового проекта»: Выбираем «Android Project»

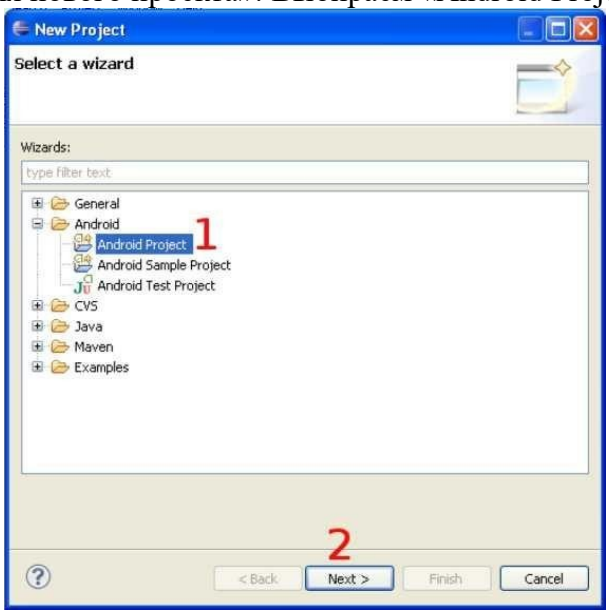


Рисунок 51

Вводим имя пакета (1) и имя класса Активности(2), который будет для нас автоматически создан мастером: Созданный проект сразу пока мы не внесли в него несовместимы с жизнью изменения является работоспособный приложением, так что мы его можем запустить:

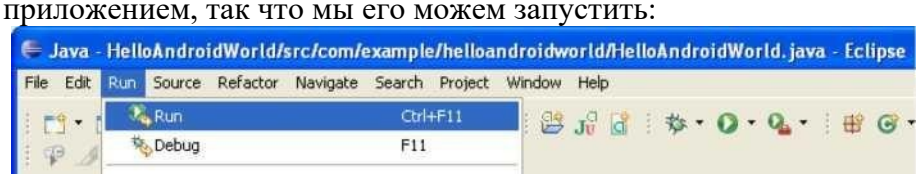


Рисунок 52

Даем понять Eclipse, как именно мы хотим запустить на выполнение наш проект:

Рисунок 53

Практическая работа № 3.6. Установка среды разработки мобильных приложений с применением виртуальной машины

Цель работы: научиться устанавливать программное обеспечение на компьютер, получить практические навыки администрирования ОС.

Установка и настройка платформы виртуализации Oracle VM VirtualBox

VirtualBox - программа абсолютно бесплатная и полностью на русском языке, что делает её очень привлекательной для использования как на домашнем, так и на рабочем компьютере. Впервые система была предоставлена в 2007 г. компанией InnoTek в двух вариантах – с открытым и закрытым исходными кодами, причем обе были бесплатны при условии некоммерческого использования. В 2008 г. платформа была перекуплена компанией Sun Microsystems, которая и занимается её разработкой в настоящее время.

Платформа представляет собой систему виртуализации для host-систем Windows, Linux и Mac OS и обеспечивает взаимодействие с гостевыми операционными системами Windows (2000/XP/2003/Vista/Seven), Linux (Ubuntu/Debian/ OpenSUSE/ Mandriva и пр.), OpenBSD, FreeBSD, OS/2 Warp.

Установка платформы Oracle VM VirtualBox

Скачать платформу, подходящую под Вашу систему, Вы можете по ссылке: http://www.virtualbox.org/wiki/Downloads

После того как установочный пакет оказался у Вас на жестком диске можно приступить к установке программы.

После запуска инсталлятора Вы увидите приветственное окно. Нажмите кнопку «Next» и в новом окне согласитесь с условиями лицензионного соглашения, поставив флажок «I accept the terms in the License Agreement». В следующем окне Вам будет предложено выбрать компоненты для установки и задать расположение исполняемых файлов. По умолчанию все компоненты устанавливаются на жесткий диск (а нам нужны все), а сама программа устанавливается в папку «Program Files» на системном диске. Если же вы хотите задать другое расположение, нажмите кнопку Browse и выберите новую папку для установки приложения.

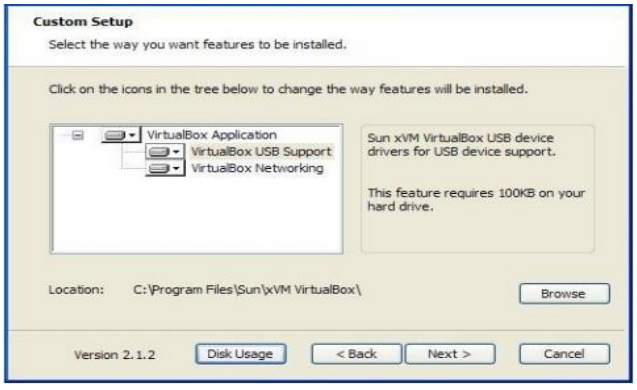


Рисунок 54

Далее процесс установки не потребует от Вас никаких вмешательств, кроме разрешения на создания ярлыков на рабочем столе и в меню «Пуск». По окончанию установки программа запуститься автоматически.

Создание и первичная настройка виртуальной машины

Запустим приложение Oracle VM VirtualBox (при установке платформы на рабочем столе создается ярлык, которым Вы можете воспользоваться). Для создания Вашей первой виртуальной машины щелкните кнопку «Создать»:

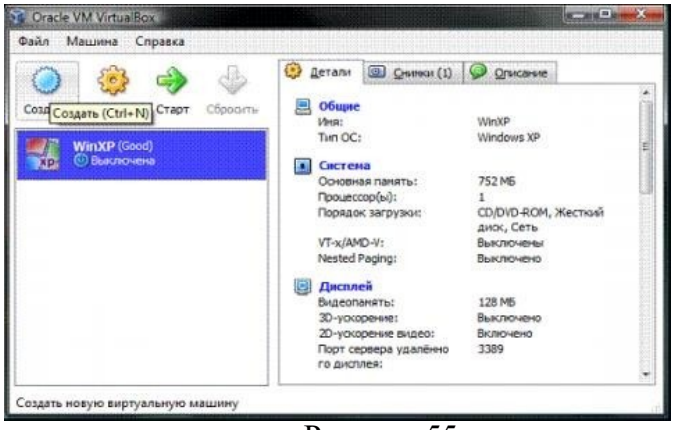


Рисунок 55

Примечание. В рассматриваемой платформе виртуализации уже существуют созданные виртуальные машины, и поэтому при первом создании своей Вы не увидите списка виртуальных операционных систем.

Перед Вами откроется новое окно, в котором будет сообщение о запуске мастера создания виртуальной машины. Нажимаем кнопку «Next» и видим новое окно, предлагающее выбрать имя операционной системы, её семейство и версию. На рис. приводится выбор Windows XP, но Вы может выбрать любую из доступных систем по своему вкусу.

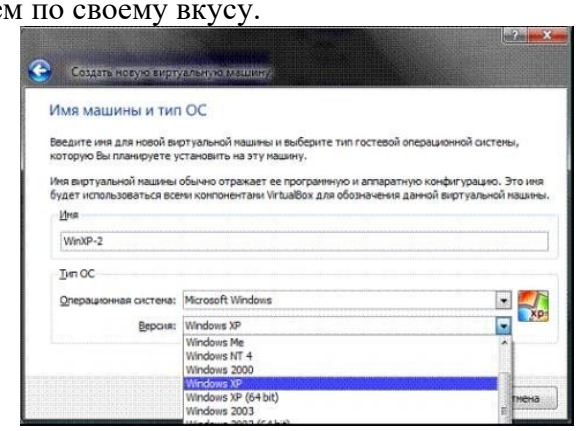


Рисунок 56

После нажатия кнопки «Next » Вам будет предложено определить размер оперативной памяти, выделяемой виртуальной машин. Здесь выбрано 1024мб, но для стабильной работы с виртуальной системой Windows XP достаточно будет и 512мб.

Далее потребуется создать виртуальный жесткий диск. Если Вы уже создавали виртуальные диски, можете использовать их, но мы рассмотрим именно процесс создания нового диска. Подтвердим, что создаваемый нами жесткий диск загрузочный, поставим флажок «Создать новый жесткий диск» и нажмем кнопку «Next».

Далее появится новое окно, которое сообщит Вам, что запущенный мастер поможет в создании виртуального диска, нажимаем кнопку «Next» для продолжения работы. В новом окне Вам будет предложено выбрать тип создаваемого диска – «динамически расширяющийся образ» или «образ фиксированного размера». Разница объясняется в справке данного окна, а от себя замечу, что непосредственно загрузочный диск удобнее создать фиксированного размера – это позволит Вам автоматически ограничить его размер, упростить и ускорить хранение, восстановление и создание резервных копий диска. К тому же, Вы можете создать для Вашей системы несколько жестких дисков

и вот уже те, которые не будут являться загрузочными, удобнее создавать динамически расширяющимися.

В следующем окне от Вас потребуется выбрать расположение создаваемого виртуального жесткого диска и его размер. Для загрузочного жесткого диска с системой Windows XP достаточно размера установленного по умолчанию (10 Гб), а вот расположить его лучше вне Вашего системного раздела, т.к. не стоит не стоит перегружать Ваш реальный загрузочный диск и создавать на нем файлы такого размера.

После этого появится окно «Итог», в котором будет указан тип, расположение и размер создаваемого Вами жесткого диска. Если Вы согласны создать диск с такими параметрами, нажмите «Финиш» и наблюдайте за процессом создания жесткого диска.

По завершения создания жесткого диска появится новое окно «Итог», в котором будут указаны параметры создаваемой Вами виртуальной машины. Если Вы не передумали ни по одному из описанных пунктов, нажимайте «Финиш» и переходите к настройке аппаратной части Вашей виртуальной машины.

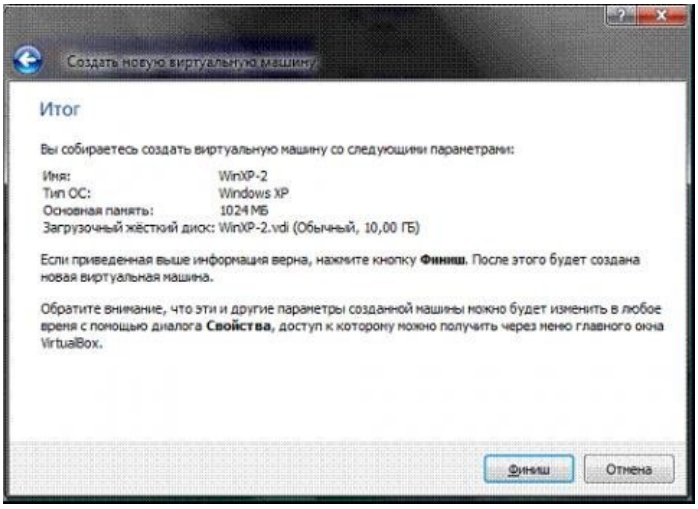


Рисунок 57 Настройка аппаратной части виртуальной машины

Итак, Вы создали виртуальный жесткий диск, теперь настала очередь собрать наш виртуальный компьютер полностью. Для этого снова вернитесь к главному окну VirtualBox, в нем Вы уже можете увидеть только что созданную виртуальную машину WinXP-2, а в поле с правой стороны представлено её описание, которое еще не похоже на описание полноценного ПК.

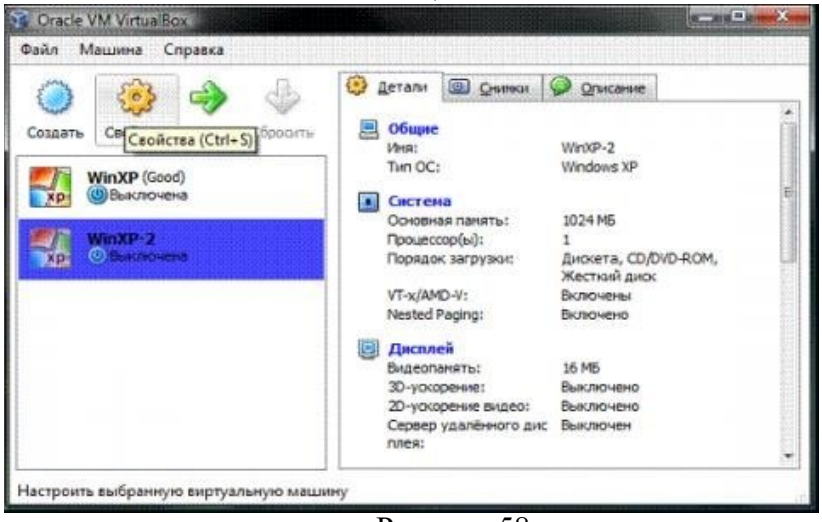


Рисунок 58

В колонке слева выберем нашу WinXP-2 и откроем её свойства, где колонка с левой стороны напоминает диспетчер устройств. На первой вкладке раздела «Общие» мы видим основные параметры нашей виртуальной машины:

Перейдем на вкладку дополнительно и посмотрим, какие настройки системы мы можем произвести:

- «Папка для снимков». Если Вы разместили Ваш жесткий в собственном расположении, то лучше и эту папку перенести туда же, т.к. снимки имеют большой вес и, опять-таки, не стоит перегружать Ваш системный диск. Моя рекомендация – создавать снимки перед каждым значительным изменением, которые Вы хотите произвести в виртуальной системе, причем даже на одну виртуальную машину Вы можете создать несколько снимков, содержащих отличные друг от друга настройки и установленные приложения;
- «Общий буфер обмена» – определение того, как будет работать буфер обмена между Вашей host-системой и виртуальной машиной. Вариантов работы буфера предоставлено четыре – «выключено», «только из гостевой ОС в основную», «только из основной ОС в гостевую», «двунаправленный». Мы выберем последний вариант, т.к. это обеспечит нам максимальное удобство в работе;
- «Сменные носители информации запоминать изменения в процессе работы», тут мы ставим флажок в знак согласия, т.к. данная опция позволит системе запомнить состояние CD\DVD- приводов;
- «Мини тулбар» – это небольшая консоль, содержащая элементы управления виртуальной машиной. Её лучше применять только в полноэкранном режиме, т.к. она полностью дублируется главным меню рабочего окна виртуальной машины. Располагать её действительно лучше сверху просто потому, что можно случайно нажать на какой-нибудь элемент управления, пытаясь, например, развернуть окно из панели задач виртуальной машины.

Перейдем к разделу система и на первой вкладке материнская плата произведем следующие настройки:

- если нужно, откорректируем размер оперативной памяти Вашей виртуальной машины, хотя окончательно убедится в правильности выбранного объема Вы сможете только после запуска виртуальной машины. Выбирать размер Вы можете, исходя из объема доступной физической памяти, установленной на Вашем ПК. Например, при наличии 4ГБ ОЗУ оптимальным будет выделение 1ГБ, т.е. одной четвертой части, что позволит Вашей виртуальной машине работать без малейших зависаний;
- откорректируем порядок загрузки - дисковод гибких дисков («дискета) можно вообще отключить, а первым обязательно поставьте CD/DVD-ROM, чтобы обеспечить возможность установки ОС с загрузочного диска. При этом в роли загрузочного диска может выступать как и компакт-диск, так и образ ISO;
- все остальные настройки описаны в динамической справке снизу, и их применение зависит от аппаратной части вашего реального ПК, причем если Вы выставите настройки неприменимые к Вашему ПК система виртуальной машины просто не запуститься;

Перейдем к вкладке «Процессор», тут Вы можете выбрать количество процессоров, установленных на Вашу виртуальную материнскую плату. Обратите внимание, что это опция будет доступна только при условии поддержки аппаратной виртуализации AMD-V или VT-x, а также включенной опции OI APIC на предыдущей вкладке.

Здесь обратите Ваше внимание на настройки аппаратной визуализации AMD-V или VT-x. Перед включением этих настроек, нужно выяснить, поддерживает ли эти возможности Ваш процессор и включены ли они по умолчанию в BIOS (нередко они отключены).

Перейдем к разделу «Дисплей». В данном разделе на вкладке «Видео» Вы можете установить размер памяти виртуальной видео карты, а также включить 2D и 3D ускорение, причем включение 2D ускорения желательно, а 3D необязательно. На вкладке «Удаленный дисплей» Вы можете включить опцию, при которой Ваша виртуальная машина будет работать как сервер удаленного рабочего стола (RDP).

Переходим к разделу носители. Тут Вы можете увидеть созданной ранее виртуальный жесткий диск и позицию с надписью пусто. Выделяем эту позицию и осуществляем настройку.

Для настройки виртуального привода компакт-дисков можно пойти двумя путями:

- первый вариант - в раскрывающемся меню «Привод» выбираем Ваш реальный или виртуальный CD/DVD-ROM (существующие в реальной системе) и загружаем в него физический диск с дистрибутивом

Windows XP или ISO-образ, если это эмулятор;

- второй вариант - щелкаем значок так, как показано на рисунке ниже и в отрывшемся окне добавляем ISO-образ загрузочного диска Windows XP, этим путем мы и пойдем.

Примечание. В данном пункте Вы уже не можете выбрать дистрибутив другой операционной системы, т.к. версия ОС уже была определена в самом начале процесса настройки виртуальной машины.

На рисунке ниже представлена процедура добавления ISO-образов в менеджер виртуальных носителей. В него Вы можете внести любое количество образов различного назначения, например, игры, дистрибутивы приложений, базы данных и пр., которые Вы сможете потом быстро переключать через главное меню окна виртуализации VirtualBox.

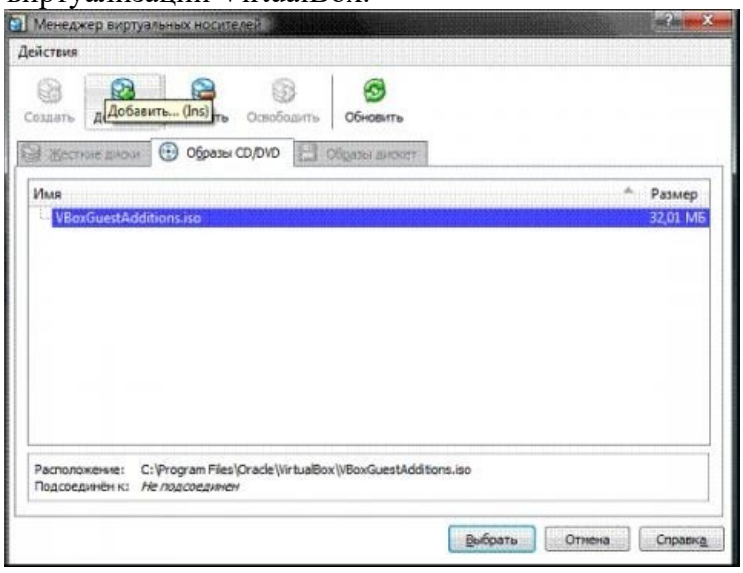


Рисунок 59

Далее Вы можете настроить слоты подключения накопителей, для упрощения описания привожу скриншоты, по которым Вы можете произвести действия по настройке. По привычке, я устанавливаю привод компакт-дисков как «Первичный мастер IDE», жесткий диск, содержащий загрузочный раздел, как «Вторичный мастер IDE», а дополнительный виртуальный жесткий диск «Первичный слейв IDE». Настройка сети и сетевого взаимодействия не рассматривается в рамках данной статьи, поэтому замечу лишь то, что сетевой адаптер типа NAT включен по умолчанию, а этого уже достаточно для предоставления Вашей виртуальной машине доступа в Интернет. Тип выбираемого адаптера должен быть «Pcnet-Fast III (Am79C973)», т.к. только для этого адаптера присутствуют драйверы в ОС Windows XP.

Раздел COM я подробно не описываю, т.к. подключать к портам данного типа уже нечего. В случае если Вам все же потребуется подключить устройство с интерфейсом RS-232C, то наиболее удобным будет включить COM-port виртуальной машины в режиме «хост-устройство», а в качестве «пути к порту» использовать имя реально порта Вашего ПК, которое Вы можете посмотреть в диспетчере устройств.

Переходим к разделу USB, здесь ставим оба доступных флажка, а затем, используя кнопку с изображением «вилки» USB и «плюса», добавляем все доступные контроллеры.

Переходим к разделу «Общие папки» и выбираем папки, которые нужно сделать доступными для виртуальной машины.

Примечание. Обратите внимание на динамическую справку снизу – именно таким образом, через окно командной строки, Вы сможете подключить общие папки к Вашей виртуальной машине.

На этом настройка аппаратной части Вашей виртуальной машины закончена, и можно перейти к установке операционной системы.

Настройка операционной системы виртуальной машины

Описание установки операционной системы в статье не описывается, т.к. на сайте представлено достаточно информации о методах и тонкостях данной операции, поэтому укажу первый шаг – возвращаемся к главному окну VirtualBox и нажимаем кнопку «Старт».

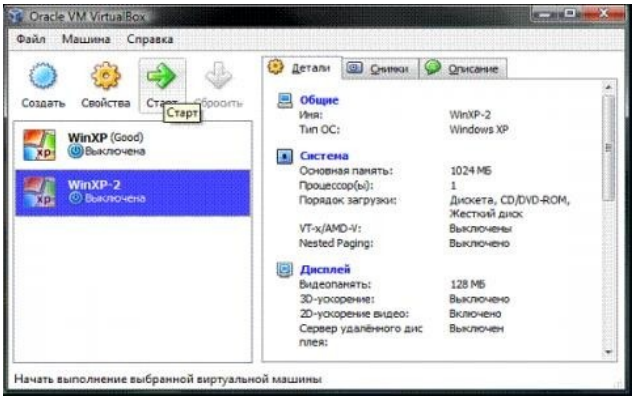


Рисунок 60

Начнется установка операционной системы Windws 7 Professional. После завершения установки Вы увидите следующее окно:

Для начала нам нужно установить драйверы для всех виртуальных аппаратных компонентов нашего виртуального ПК. Для этого в главном меню выбираем пункт «Устройства» - «Приводы оптических дисков» - «VboxGuestAdditions.iso». Впоследствии таким же образом Вы сможете подключить к своей виртуальной машине физический CD-ROM или загрузить ISO-образ.

После подключения образа «VboxGuestAdditions.iso» в папке Мой компьютер Вы увидите, что в привод компакт-дисков загружен данный виртуальный диск – остается его только запустить двойным щелчком левой кнопки мыши.

Сам процесс установки происходит практически без Вашего участия и только в случае, если Вы включили ранее 3D-ускорение, то следует выбрать соответствующий компонент для дополнительной установки.

В последнем окне процесса инсталляции Вам будет предложено перезагрузить виртуальную машину - соглашайтесь, после перезагрузки Вы увидите, что изображение стало четче, размер окна виртуальной машины изменяется динамически, включена функция интеграции мыши и есть доступ в Интернет.

Давайте теперь подключим общие папки, чтобы получить возможность перенести в созданную виртуальную машину нужные для работы файлы и установить приложения. Это можно сделать с помощью командной строки, следуя справке VirtualBox, но я приведу способ с использованием проводника Windows. Для этого откройте папку Мой компьютер, в главном меню выберите «Сервис» - «Подключить сетевой диск» и открывшемся окне в поле папка введите \\vboxsrv\имя_общей_папки, т.е. в нашем случае: \\vboxsrv\Share

После этих действий в папке «Мой компьютер» появится Ваша общая папка, доступная в качестве сетевого диска.

Давайте теперь проверим, есть ли у Вас доступ в Интернет. Для этого откройте: меню «Пуск» - «Программы» - «Стандартные» - «CMD-командная строка» и в открывшемся окне (рис. 31) введите следующую команду: Ping ya.ru

Если в результате отработки команды Вы видите, что пакеты отправлены и получены, пусть даже частично, то у Вас все получилось, и доступ в Интернет виртуальной машине обеспечен.

Быстрый доступ и комфортная работа

Далее хотелось бы сказать несколько слов о двух возможностях VirtualBox, которые помогут ускорить Вашу работу с виртуальными машинами и сделать ее комфортнее.

Ярлык для быстрого запуска виртуальной машины

Для более быстрого и удобного запуска Вашей виртуальной машины создадим ярлык именно для нее. Сделать это можно следующим образом:

a. Щелчком правой кнопки мыши на рабочем столе вызовем контекстное меню и выберем пункт «Создать ярлык»;
b. В открывшемся окне в поле «Укажите размещение объекта» введите "C:\Program Files\Oracle\ VirtualBox\VBoxManage.exe"
c. startvm Win7Pro;
d. В следующем окне введите имя ярлыка, например, «Win7Pro» и нажмите кнопку «Готово»;

e. На Вашем рабочем столе появился созданный ярлык «Win7Pro», щелкните на нем правой кнопкой;
f. В открывшемся контекстном меню выберите «Свойства»;
g. В появившемся окне выберите сменить значок и поле выбора значков выберите любой значок, который посчитаете нужным.

Режим интеграции дисплеев

В режиме интеграции дисплеев Вы легко можете со своего рабочего стола организовать доступ к рабочему столу и элементам управления виртуальной машины, т.е. все окна, открываемые Вами в виртуальной машине, будут отображаться уже на Вашем рабочем столе, а не в отдельном окне VirtualBox. Данная функция значительно облегчает доступ к виртуальной машине, её элементам управления и, установленным на ней приложениям. Чтобы включить этот режим Вам нужно в главном меню окна визуализации VirtualBox выбрать пункт с соответствующим названием или нажать сочетание клавиш «HOST + L», где «HOST» клавиша – левый «Ctrl» (по умолчанию).

Программа выполнения работы

1. Скачайте платформу, подходящую под Вашу систему по ссылке: http://www.virtualbox.org/wiki/Downloads
2. Создайте на диске D: папку DISTRIB. С указанного преподавателем сетевого ресурса скачайте в эту папку ISO-образ операционной системы Windows 7 Pro (32 bit).
3. Запустите инсталлятор и выполните установку платформы Oracle VM VirtualBox.
4. Создайте новую виртуальную машину Win7Pro. Выберите для новой виртуальной машины объем оперативной памяти 1024 MB и создайте новый загрузочный жесткий диск фиксированного размера емкостью 25 GB на несистемном диске (диске D:) вашего компьютера.
5. Выполните настройку аппаратной части созданной виртуальной машины.
6. На несистемном диске (диске D:) вашего компьютера создайте папку для снимков.
7. Настройте буфер обмена между хостовой и виртуальной машинами как «двунаправленный».
8. Откорректируйте порядок загрузки. Первым обязательно поставьте CD/DVD-ROM, чтобы обеспечить возможность установки ОС с загрузочного диска. При этом в роли загрузочного диска может выступать как и компакт-диск, так и образ ISO.
9. На вкладке «Процессор» выберите количество процессоров, установленных на Вашу виртуальную материнскую плату. Обратите внимание, что это опция будет доступна только при условии поддержки аппаратной виртуализации AMD-V или VT-x, а также включенной опции OI APIC на предыдущей вкладке. Здесь обратите Ваше внимание на настройки аппаратной визуализации AMD-V или VT-x. Перед включением этих настроек, нужно выяснить, поддерживает ли эти возможности Ваш процессор и включены ли они по умолчанию в BIOS (нередко они отключены).
10. В разделе «Дисплей» на вкладке «Видео» установите размер памяти виртуальной видеокарты, а также включите 2D и 3D ускорение, причем включение 2D ускорения желательно, а 3D необязательно.
11. В разделе “Носители” выделите позицию с надписью “Пусто” и добавьте ISO-образ загрузочного диска Windows 7.
12. В разделе USB поставьте оба доступных флажка, а затем, используя кнопку с изображением «вилки» USB и «плюса», добавьте все доступные контроллеры.
13. Запустите установку и установите операционную систему Windows 7 Pro.
14. Установите драйверы для всех виртуальных аппаратных компонентов нашего виртуального ПК. Для этого в главном меню выберите пункт «Устройства» - «Приводы оптических дисков» - «VboxGuestAdditions.iso». Впоследствии таким же образом Вы сможете подключить к своей виртуальной машине физический CD-ROM или загрузить ISO-образ. После подключения образа «VboxGuestAdditions.iso» в папке Мой компьютер Вы увидите, что в привод компакт-дисков загружен данный виртуальный диск – остается его только запустить двойным щелчком левой кнопки мыши.
15. Создайте ярлык для более быстрого и удобного запуска Вашей виртуальной машины.
16. Щелчком правой кнопки мыши на рабочем столе вызовите контекстное меню и выберите пункт «Создать ярлык»;
17. В открывшемся окне в поле «Укажите размещение объекта» введите "C:\Program Files\Oracle\

VirtualBox\VBoxManage.exe"

18. startvm Win7Pro;
19. В следующем окне введите имя ярлыка, например, «Win7Pro» и нажмите кнопку «Готово»;
20. На Вашем рабочем столе появился созданный ярлык «Win7Pro», щелкните на нем правой кнопкой;
21. В открывшемся контекстном меню выберите «Свойства»;
22. В появившемся окне выберите значок? Который посчитаете нужным.
23. Нажмите кнопку «ОК», а затем «Применить».
24. Включите режим интеграции дисплеев, выбрав в Главном меню пункт с соответствующим названием.
25. Оформите отчет по практической работе.

Практическая работа № 3.7. Создание эмуляторов. Подключение устройств. Настройка режима терминала

Цель работы: Знакомство с интерфейсом среды программирования. Изучение структуры проекта Рассмотрите основные понятия Android проекта.

Структура проекта

- src—«исходный код» приложения (java-классы)
- assets —пустая директория. Может использоваться для сохранения raw-файлов.
- gen—хранилище генерируемых системных файлов. В частности, здесь располагается файл R.java, в котором хранятся идентификаторы всех ресурсов, создаваемых в проекте (строковые ресурсы и т.п.).
- libs—различные библиотеки, используемые приложением
- res—ресурсы проекта.
- AndroidManifest.xml—файл описания проекта (поддерживаемые версии SDK, версия приложения и т.п.)
  - project.properties—файл, включающий настройки проекта, такие как

build target. Ресурсы проекта

- anim/ Содержит XML файлы, компилируемые в анимационные объекты.
- color/ Содержит XML файлы описывающие цвета.
- drawable/ Содержит растровые файлы (PNG, JPEG, orGIF), 9-Patch файлы, и XML файлы, описывающие Drawableshapes или Drawableobjects включающие
- множественные состояния(нормальное, нажатое, состояние фокуса).
- layout/ Содержит XML файлы описывающие макеты экрана
- menu/ Содержит XML файлы, определяющие меню приложения.
- raw/ Для хранения произвольных файлов.
- values/ Содержит XML файлы, компилируемые во множество видов ресурсов (strings.xmlи т.д).
- xml/ СодержитXML файлы конфигурирующие компоненты приложения. Например, XML файл определяющий экран настроек.

Пример простейшего файла AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.untitled"
android:versionCode="1
android:versionName="1.0
">
<uses-sd kandroid:minSdkVersion="19"/>
<application

android:label="@string/app_name" android:icon="@drawable/ic_launcher">
<activity
android:name="MyActivity"
```

```
android:label="@string/app_name
">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
```

Для начала работы требуется:

- Java Development Kit
- Android Software Development

Kit Android Virtual Device Manager

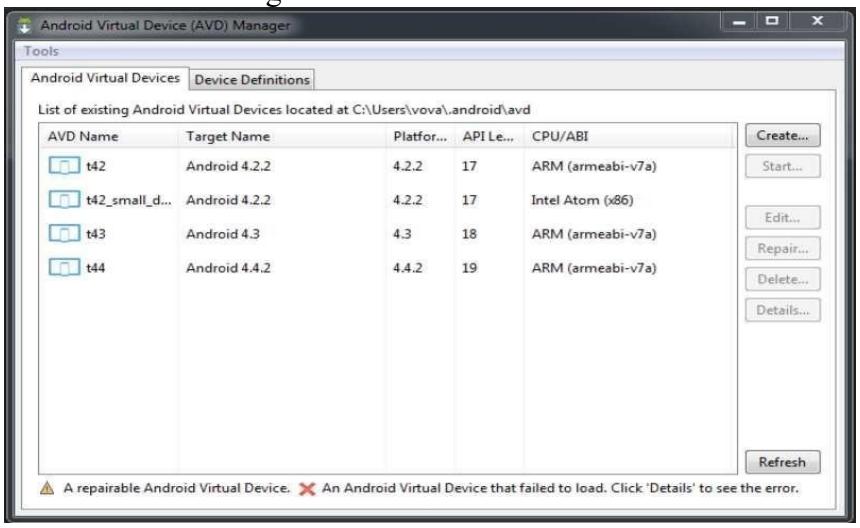


Рисунок 61

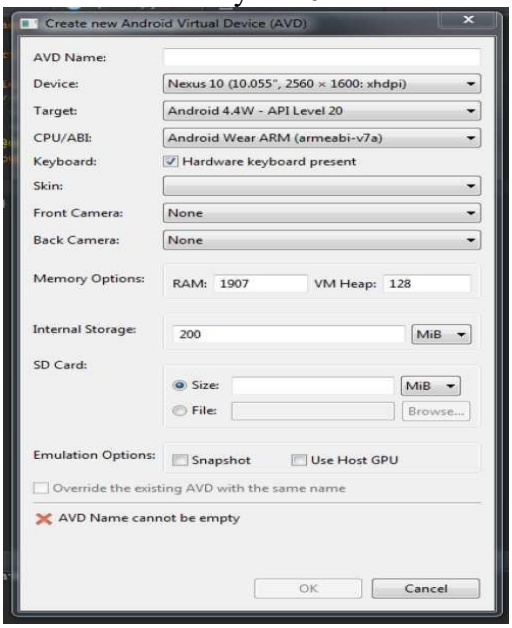


Рисунок 62

Задание 1. Создание проекта приложения

Запустите среду программирования в IDE InelliJ Idea

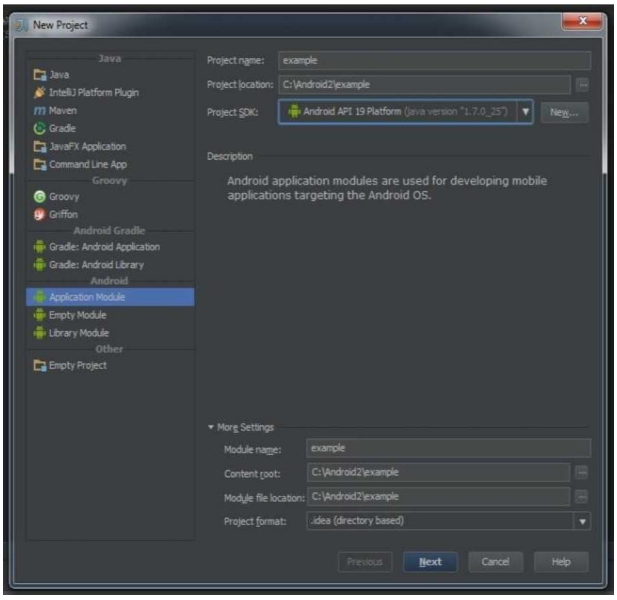


Рисунок 63

Задание 2. Создание приложений с одним экраном (Activity) Необходимо создать два activity и организовать переход между ними. Содержимое Activity 1 кнопка с именем btnl

Содержимое Activity 2: TextView с текстом «Переданный параметр: значение_параметра». значение_параметра - принятый параметр из Activity 1.

При запуске приложения пользователь должен попадать на экран с Activity 1. После нажатия на кнопку btn1 необходимо осуществить переход к Activity 2 и передавать параметр из Activity 1. В качестве значения передаваемого параметра использовать свою фамилию.

Ключевые классы: Activity, Intent, Button, TextView.

Практическая работа № 3.8. Создание нового проекта. Изучение кода. Комментирование кода. Изменение элементов дизайна

Цель работы: Изучить работу Android приложения с базой данных. SQLite

SQLite—компактная встраиваемая реляционная база данных с открытым исходным кодом. Поддерживает динамическое типизированиеданных.

Возможные типы полей: INTEGER, REAL, TEXT, BLOB. SQLiteOpenHelper

Абстрактные методы

onCreate( ) -вызывается при первом создании базы данных;

onUpgrade( ) -вызывается при модификации базы данных Реализация метода onCreate

```
public void onCreate( SQLiteDatabasedb) { db.execSQL( "CREATE TABLE"+ TABLE_NAME
+ "(_id INTEGER PRIMARY KEYAUTOINCREMENT ,"
+ COL NAME + " TEXT ,"+ COL PHONE + " TEXT ) ; " ) ;
```

Реализация метода onUpgrade @ Override

```
publicvoid onUpgrade( SQLiteDatabasedb, intoldVersion, intnewVersion)
{
db.execSQL( "DROP TABLE IF EXISTS"+ TABLE_NAME ) ;
onCreate( db) ;
}
```

Чтение из базы данных

```
Cursor query( String table, String [ ] columns, String selection, String [ ] selectionArgs,
        String groupBy, String having , String sortOrder)
```

Позиционирование курсора

1. moveToFirst( ) -перемещает курсор в первую запись в выборке;

2. moveToLast( ) -перемешает курсор в последнюю запись в выборке;
3. moveToNext( ) -перемешает курсор в следующую запись и одновременноопределяет, существует ли эта запись. Метод moveToNext( ) возвращаетЦие, если курсор указывает на другую строку после перемещения, и false^enM текущая запись была последней в выборке;
4. moveToPrevious( ) -перемешает курсор в предыдущую запись;
5. moveToPosition( ) -перемешает курсор в указанную позицию;
6. getPosition( ) -возвращает текущий индекс позиции курсора.
7. isFirst( ) ;
8. isLast( ) ;
9. isBeforeFirst( ) ;
10. isAfterLast( ) . Обновление и удаление записей
11. intupdate(String table, ContentValuesvalues, String whereClause, String [ ] whereArgs)
12. intdelete (String table, String whereClause, String [ ] whereArgs)

Задание 1. Необходимо создать приложение, взаимодействующее с базой данных. Первое активити должно содержать три кнопки. При нажатии на первую кнопку должно открываться новое активити, выводящее информацию из таблицы «Одногруппники» в удобном для восприятия формате.

При запуске приложения необходимо:

1. Создать БД, если ее не существует.
2. Создать таблицу «Одногруппники», содержащую поля:
3. ID;
4. ФИО;
5. Время добавления записи.
6. Удалять все записи из БД, а затем вносить 5 записей об одногруппниках. При нажатии на вторую кнопку необходимо внести еще одну запись в таблицу.

При нажатии на третью кнопку необходимо заменить ФИО в последней внесенной записи на Иванов Иван Иванович.

Задание 2. Создать новое отдельное приложение на основе приложения, созданного в части 1. Переопределить функцию onUpgrade. При изменении версии БД необходимо удалить таблицу«Одногруппники», создать таблицу «Одногруппники» содержащую следующие поля:

1. ID;
2. Фамилия;
3. Имя;
4. Отчество;
5. Время добавления записи.
6. Изменить версию базы данных.

Задание 3.

1. Создайте новый проект MetroPicker.
2. Добавьте вспомогательную Активность ListViewActivity для отображения и выбора станций метро, в качестве заготовки используйте результаты лабораторной работы «Использование ListView» .
3. Отредактируйте файл разметки res/layout/main.xml: добавьте кнопку выбора станции метро, присвоив идентификаторы виджетам TextView и Button для того, чтобы на них можно было ссылаться в коде.
4. Установите обработчик нажатия на кнопку в главной Активности для вызова списка станции и выбора нужной станции.
5. Напишите нужный обработчик для установки выбранной станции метро в виджет TextView родительской Активности (метод setText виджета TextView позволяет установить отображаемый текст). Не забудьте обработать ситуацию, когда пользователь нажимает кнопку «Назад» (в этом случае «никакой станции не выбрано» и главная Активность должна известить об этом пользователя).
6. Убедитесь в работоспособности созданного приложения, проверив реакцию различные действия

потенциальных пользователей.
Неявные намерения

7. Неявные намерения используются для запуска Активностей для выполнения заказанных действий в условиях, когда неизвестно (или безразлично), какая именно Активность (и из какого приложения) будет использоваться.
8. При создании Намерения, которое в дальнейшем будет передано методу startActivity, необходимо назначить действие (action), которое нужно выполнить, и, возможно, указать URI данных, которые нужно обработать. Также можно передать дополнительную информацию с помощью свойства extras Намерения. Android сам найдет подходящую Активность (основываясь на характеристиках Намерения) и запустит ее. Пример неявного вызова телефонной «звонилки»:
9. Intent intent = new Intent(Intent.ACTION DIAL, Uri.parse("tel:(495)502-99-11"));
10. startActivity(intent);
11. Для определения того, какой именно компонент должен быть запущен для выполнения действий, указанных в Намерениях, Android использует Фильтры Намерений (Intent Filters'). Используя Фильтры Намерений, приложения сообщают системе, что они могут выполнять определенные действия (action) с определенными данными (data) при определенных условиях (category) по заказу других компонентов системы.
12. Для регистрации компонента приложения (Активности или Сервиса) в качестве потенциального обработчика Намерений, требуется добавить элемент <intent-filter> в качестве дочернего элемента для нужного компонента в Манифесте приложения. У элемента <intent-filter> могут быть указаны следующие дочерние элементы (и соответствующие атрибуты у них):
13. <action>. Атрибут android:name данного элемента используется для указания названия действия, которое может обслуживаться. Каждый Фильтр Намерений должен содержать не менее одного вложенного элемента <action>. Если не указать действие, ни одно Намерение не будет «проходить» через этот Фильтр. У главной Активности приложения в Манифесте должен быть указан Фильтр Намерений с действием android.intent.action.MAIN
14. <category>. Сообщает системе, при каких обстоятельствах должно обслуживаться действие (с помощью атрибута android:name). Внутри <intent-filter> может быть указано несколько категорий. Категория android.intent.category.LAUNCHER требуется Активности, которая желает иметь «иконку» для запуска. Активности, запускаемые с помощью метода startActivity, обязаны иметь категорию android.intent.category.DEFAULT
15. <data>. Дает возможность указать тип данных, которые может обрабатывать компонент. <intent-filter> может содержать несколько элементов <data>. В этом элементе могут использоваться следующие атрибуты:
16. android:host : имя хоста (например, www.specialist.ru)
17. android:mimetype : обрабатываемый тип данных (например, text/html)
18. android:path : «путь» внутри URI (например, /course/android)
19. android:port : порт сервера (например, 80)
20. android:scheme : схема URI (например, http)

Пример указания Фильтра Намерений:

```
<activity android:name=".MyActivity" android:label="@string/app name" > <intent-filter>
<action android:name="android.intent.action.SEND" />
<category android:name="android.intent.category.DEFAULT" />
<data android:mimeType="text/plain" />
</intent-filter>
</activity>
```

21. При запуске Активности с помощью метода startActivity неявное Намерение обычно подходит только одной Активности. Если для данного Намерения подходят несколько Активностей, пользователю предлагается список вариантов.
22. Определение того, какие Активности подходят для Намерения, называется Intent Resolution. Его задача - определить наиболее подходящие Фильтры Намерений, принадлежащие компонентам установленных приложений. Для этого используются следующие проверки в указанном порядке:

23. Проверка действий. После этого шага остаются только компоненты приложений,у которых в Фильтрах Намерений указано действие Намерения. В случае, если действие в Намерении отсутствует, совпадение происходит для всех Фильтров Намерений, у которых указано хотя бы одно действие.

24. Проверка категорий. Все категории, имеющиеся у Намерения, должны присутствовать в Фильтре Намерений. Если у Намерения нет категорий, то на данном этапе ему соответствуют все Фильтры Намерений, за одним исключением, упоминавшимся выше: Активности, запускаемые с помощью метода startActivity, обязаны иметь категорию android.intent.category.DEFAULT, так как Намерению, использованному в этом случае, по умолчанию присваивается данная категория, даже если разработчик не указал ничего явно. Из этого исключения, в свою очередь, есть исключение: если у Активности присутствуют действие android.intent.action.MA.IN и категория android.intent.category.LAUNCHER, ему не требуется иметь категорию android.intent.category.DEFAULT.

25. Проверка данных. Здесь применяются следующие правила:

Намерение, не содержащее ни URI, ни типа данных, проходит через Фильтр,если он тоже ничего перечисленного не содержит.

Намерение, которое имеет URI, но не содержит тип данных (и тип данных невозможно определить по URI), проходит через Фильтр, если URI Намерения совпадает с URI Фильтра. Это справедливо только в случае таких URI, как mailto: или tel:, которые не ссылаются на реальные данные.

Намерение, содержащее тип данных, но не содержащее URI подходит толькодля аналогичных Фильтров Намерений.

Намерение, содержащее и тип данных, и URI (или если тип данных может быть вычислен из URI), проходит этот этап проверки, только если его тип данных присутствует в Фильтре. В этом случае URI должен совпадать, либо(!) у Намерения указан URI вида content: или file:, а у Фильтра URI не указан. То есть, предполагается, что если у компонента в Фильтре указан только тип данных, то он поддерживает URI вида content: или file:.В случае, если после всех проверок остается несколько приложений, пользователю предлагается выбрать приложение самому. Если подходящих приложений не найдено, в выпустившей Намерение Активности возникает Исключение.

Задание5.

Модифицируйте проект MetroPicker следующим образом:

1. Добавьте главное меню в Активность, отображающую список станций метро. В меню должен быть один пункт: «вернуться». Меню создайте динамически в коде,без использования строковых ресурсов.
2. Динамически создайте контекстное меню для Представления TextView, отображающего выбранную станцию метро главной Активности. Выбор пункта меню должен сбрасывать выбранную станцию.
3. Для главной Активности создайте основное меню из двух пунктов: «сбросить» и «выйти». Реализуйте нужные функции при выборе этих пунктов.
4. Повторите реализацию п.п. 1, 2 и 3 с помощью ресурсов, описывающих меню. Задание 6.
1. Создайте новый Android проект ListViewSample.
2. В каталоге res/values создайте файл arrays.xml с данными полученными у преподавателя.
3. В каталоге res/layout создайте файл list_item.xml с данными полученными у преподавателя.
4. Модифицируйте метод onCreate вашей Активности:

```
@Override public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedlnstanceState);
Resources r = getResources();
String[] stationsArray = r.getStringArray(R.array.stations);
ArrayAdapter<String> aa = new ArrayAdapter<String>(this, R.layout.list item, stationsArray);
setListAdapter(aa); ListView
```

lv = getListView();
}
5. Измените базовый класс Активности с Activity на ListActivity.
6. Запустите приложение.
7. Для реакции на клики по элементам списка требуется добавить обработчик такогособытия, с помощью метода setOnItemClickListener. В качестве обработчика будет использоваться анонимный объект класса OnItemClickListener. Добавьте следующий код в нужное место:

```
lv.setOnItemClickListener(new OnItemClickListener() {
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
CharSequence text = ((TextView) v).getText(); int duration = Toast.LENGTH_LONG; Context context = getApplicationContext();
Toast.makeText(context, text, duration).show();
}
});
```

8. Запустите приложение и «покликайте» по станциям метро.

Задание 7. «Использование управляющих элементов в пользовательском интерфейсе» Подготовка

1. Создайте новый проект ControlsSample.
2. Отредактируйте файл res/layout/main.xml так, чтобы остался только корневой элемент LinearLayout. В него в дальнейшем будут добавляться необходимые дочерние элементы:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout width="fill parent" android:layout height="fill parent" android:orientation="vertical" >
</LinearLayout>
```

Использование графической кнопки

Для использования изображения вместо текста на кнопке потребуются три изображения для трех состояний кнопки: обычное, выбранное («в фокусе») и нажатое. Все эти три изображения с соответствующими состояниями описываются в одном XML файле, который используется для создания такой кнопки.

1. Скопируйте нужные изображения кнопки в каталог res/drawable-mdpi, для обновления списка содержимого каталога в Eclipse можно использовать кнопку F5.
2. В этом же каталоге создайте файл smile_button.xml, описывающий, какие изображения в каких состояниях кнопки нужно использовать:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/smile_pressed" android:state_pressed="true"/>
<item android:drawable="@drawable/smile_focused" android:state_focused="true"/>
<item android:drawable="@drawable/smile_normal" />
</selector>
```

3. Добавьте элемент Button внутри LinearLayout в файле разметки res/layout/main.xml:

```
<Button
android:id="@+id/butto
n"
android:layout_width="wrap_content" android:layout_height="wrap_content" android:background="@drawable/smile_button" android:onClick="onButtonClicked" android:padding="10dp" />
```

4. Обратите внимание на атрибут android:onClick= "onButtonClicked", указывающий, какой метод из Активности будет использоваться как обработчик нажатия на данную кнопку. Добавьте этот метод в Активность:

```
public void onButtonClicked(View v) {
Toast. makeText (this, "Кнопка нажата", Toast. LENGTH_SHORT).show();
}
```

5. Запустите приложение и посмотрите, как изменяется изображение кнопки в

разныхсостояниях, а также как функционирует обработчик нажатия на кнопку.

Использование виджета CheckBox

1. Добавьте элемент CheckBox внутри LinearLayout в файле разметки res/layout/main.xml:

```
<CheckBox
android:id= "@+id/checkbox"
android:layout_width="wrap_content"
android:layout_height="wrap_content" android:onClick="onCheckboxClicked"
android:text="Выбери меня" />
```

2. Атрибут android:onClick= "onCheckboxClicked" определяет, какой метод из Активности будет использоваться как обработчик нажатия на виджет. Добавьте этот метод в Активность:

```
public void onCheckboxClicked(View v) { if (((CheckBox)
v).isChecked()) { Toast.makeText(this, "Отмечено",
Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(this, "Не отмечено", Toast.LENGTH_SHORT).show();
}
}
```

3. Запустите приложение и посмотрите на поведение чекбокса в разных ситуациях. Использование виджета ToggleButton

Данный виджет хорошо подходит в качестве альтернативы радиокнопкам и чекбоксам, когда требуется переключаться между двумя взаимоисключающими состояниями, например, Включено/Выключено.

1. Добавьте элемент ToggleButton внутри LinearLayout в файле разметки res/layout/main.xml:

```
<ToggleButton android:id= "@+id/togglebutton" android:layout_width= "wrap_content"
android:layout_height="wrap_content" android:textOn="Звонок включен" android:textOff="Звонок
выключен" android:onClick= "onToggleClicked"/>
```

2. Атрибут android:onClick= "onToggleClicked" определяет, какой метод ИЗ Активности будет использоваться как обработчик нажатия на виджет. Добавьте этот метод в Активность:

```
public void onToggleClicked(View
v) { if (((ToggleButton)
v).isChecked()) {
Toast.makeText(this, "Включено", Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(this, "Выключено", Toast.LENGTH_SHORT).show();
}
}
```

3. Запустите приложение и проверьте его функционирование. Использование виджета RadioButton

Радиокнопки используются для выбора между различными взаимоисключающими вариантами. Для создания группы радиокнопок используется элемент RadioGroup, внутри которого располагаются элементы RadioButton.

1. Добавьте следующие элементы разметки внутри LinearLayout в файле res/layout/main.xml:

```
<RadioGroup
android:layout_width="fill_parent"
android:layout_height="wrap_conte
nt" android:orientation="vertical" >
<RadioButton
android:id= "@+id/radio_dog"
```

```
android:layout_width= "wrap_content" android:layout_height=
"wrap_content" android:onClick="onRadioButtonClicked"
android:text="Собачка" />
<RadioButton
android:id="@+id/radio_cat"
android:layout_width="wrap_content"

android:layout_height="wrap_content" android:onClick="onRadioButtonClicked" android:text="Кошечка" />
<RadioButton
android:id= "@+id./radio_rabbit" android:layout_width="wrap_content" android:layout
height="wrap content"
android:onClick="onRadioButtonClicked
android:text="Кролик" />
</RadioGroup>
```

2. Добавьте метод onRadioButtonClicked в Активность:

```
public void onRadioButtonClicked(View v) { RadioButton rb =
(RadioButton) v; Toast. makeText(this, "Выбрано животное: " +
rb.getText(), Toast.LENGTH_SHORT).show();
}
```

3. Проверьте работу приложения. Использование виджета EditText

Виджет EditText используется для ввода текста пользователем. Установленный для этого виджета обработчик нажатий на кнопки будет показывать введенный текст с помощью Toast.

1. Добавьте элемент EditText внутри LinearLayout в файле разметки res/layout/main.xml:

```
<EditText
android:id= "@+id/user_name"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:hint="Введите имя"/>
```

2. Для обработки введенного пользователем текста добавьте следующий код в конце метода onCreate. Обратите внимание, этот обработчик, в отличие от предыдущих, использованных нами, возвращает значение true или false. Семантика этих значений традиционна: true означает, что событие (event) обработано и больше никаких действий не требуется, false означает, что событие не обработано этим обработчиком и будет передано следующим обработчикам в цепочке. В нашем случае реагирование происходит только на нажатие (ACTION_DOWN) кнопки Enter (KEYCODE_ENTER):

```
final EditText userName = (EditText) findViewById(R.id.user
name); userName.setOnKeyListener(new
View.OnKeyListener() { @Override
public boolean onKey(View v, int keyCode, KeyEvent event) {
if ((event.getAction() == KeyEvent.ACTION DOWN)
&& (keyCode == KeyEvent.KEYCODE_ENTER)) {
Toast.makeText(getApplicationContext(), userName.getText(),
Toast.LENGTH_SHORT).show();
return true;
}
return false;
}
});
```

3. Запустите приложение и проверьте его работу.
4. Добавьте кнопку «Очистить» в разметку и напишите обработчик, очищающий текстовое поле (используйте метод setText виджета EditText)
5. Проверьте работу приложения.

Практическая работа № 3.9. Обработка событий: подсказки. Обработка событий: цветовая индикация. Подготовка стандартных модулей. Обработка событий: переключение между экранами

Цель работы: Создание обработчиков событий. Изучить работу с потоками. Научиться работать с мультимедиа файлами. Изучить работу с классом AsyncTask

Главный и обычный потоки Асинхронные потоки в Android

AsyncTask-это специальный абстрактный класс, предоставляющий набор методов для реализации:

- onPreExecute-для размещения инициализирующего кода (UI поток)
- doInBackground-для размещения тяжелого кода, который будет выполняться в другом потоке
- onProgressUpdate-для информирования о прогрессе (UI поток)
- onPostExecute-для обработки результата, возвращенного doInBackground(UI поток) и вспомогательных методов
- isCancelled-для получения информации об отмене задачи
- publishProgress-для перевода сообщения о прогрессе в UI поток с последующим вызовом onProgressUpdate

Последовательность выполнения методов AsyncTask onPreExecute doInBackground onPostExecute publishProgress onProgressUpdate

Правила использования AsyncTask:

Объект AsyncTaskдолжен быть создан в UI-потоке

- Метод execute должен быть вызван в UI-потоке
- Метод executeможет быть запущен только один раз
- Не вызывайте методы onPreExecute, doInBackground, опРоз1Execи1еи onProgressUpdate Передача данных в AsyncTask

Объявляем класс

```
Class MyAsyncTaskextends AsyncTask<String, Integer, Long>{
}
```

- Первый параметр используется методом doInBackground protected-Long doInBackground(String... urls)
- Второй параметр используется методом onProgressUpdate protectedvoidonProgressUpdate(Integer... progress)
- Третий параметр используется методом onPostExecute protectedvoidonPostExecute(Longresult)

Промежуточные данные

Последовательность действий для передачи промежуточных данных в основной поток программы:

- В методе doInBackgroundвызываем метод publishProgress
- В методе onProgressUpdate обрабатываем переданный в publishProgress параметр и выводим прогресс

Метод get

- Возвращает результат выполнения метода doInBackground
- Вызывается из UI потока

```
MyAsyncTaskat =
newMyAsyncTask(); result=
at.get();
```

Задание 1. Разработать мобильное приложение, состоящее из четырех activity.

После запуска приложения пользователь должен попадать на экран с activityt. На данном экране должно быть представлено меню, состоящее из четырех кнопок. Высота кнопок должна составлять 20% от высоты экрана. Расстояние между кнопками - 2%. Первая и последняя кнопка должны быть на равном расстоянии от краев экрана. Ширина кнопок 75%, выравнивание посередине.

После нажатия на первую кнопку пользователь должен переходить к activity2, его внешний вид представлен на рисунке 64. Верстка должна осуществляться с использованием LinearLayout, ширина кнопок должна задаваться в процентах от ширины экрана.

Рисунок 64

После нажатия на вторую кнопку в activity1 пользователь должен переходить к activity3, его внешний вид представлен на рисунке 65. Верстка должна осуществляться с использованием RelativeLayout (не использовать LinearLayout).

Рисунок 65

Третья кнопка в activity1 должна создавать activity3. Внешний вид activity3 представлен на рис.66

Рисунок 66

Кнопка должна быть выровнена по центру экрана. Цвет обводки кнопки #505050. Толщина обводки в соответствии с месяцем вашего рождения (от 1 до 12). Радиус скругления 24dp. Цвет фона экрана #FFFFFF. При нажатии на кнопку ее цвет должен изменяться на светло-зеленый.

Нажатие на четвертую кнопку в activity1 должно приводить к закрытию приложения. Задание 2. Рассмотрите пример передачи данных. MyAsyncTaskat = newMyAsyncTask(); at.execute("url1", "url2");
doInBackground(String... urls)
Задание 3. Рассмотрите пример вывода промежуточных данных. @Override
Protected void doInBackground(String... urls) {
try{ int cnt= 0;
for(Stringurl: urls) {
// обрабатываем первый параметр
// выводим промежуточные результаты cnt++;
publishProgress(cnt);
}
TimeUnit.SECONDS.sleep(l);
}
catch(InterruptedExceptione)
{ e.printStackTrace();
}
return null;
}
@Override
protected void onProgressUpdate(Integer... values) {
super.onProgressUpdate(values); tv.setText("o6pa6oiaHO " + values[0] + " параметров");
}
Задание 4. Проверьте пример создания простой асинхронной задачи public class MainActivity extends Activity{
MyAsyncTask at; TextView tv;
public void onCreate(Bundle savedInstance State) {
super.onCreate(savedInstanceState); setContentView(R.layout.main);
tv= (TextView) findViewById(R.id.tv);
MyAsyncTask at = new MyAsyncTask(); at.execute();
}
}

Задание 5. На основании изученных примеров разработать приложение сохраняющее статистику проигрываемых песен на радио Мегабайт. Для сохранения песни и названия необходимо создать базу данных, содержащую таблицу со следующими полями:

- ID
- Исполнитель
- Название трека
- Время внесения записи

При включении приложения необходимо производить проверку подключения к Интернету. В случае если подключение отсутствует - выводить всплывающее сообщение (Toast) с предупреждением о запуске в автономном режиме (доступен только просмотр внесенных ранее записей).

После включения приложение должно производить асинхронный опрос сервера с интервалом 20 секунд. Если название трека не совпадает с последней записью в таблице необходимо произвести запись в БД.

URL адрес, по которому можно получить информацию о текущем треке и исполнителе: http://media.ifmo.ru/api get current song.php
Формат возвращаемых данных - JSON.
В случае успешного выполнения запроса результат будет иметь вид:

{"result": "success", "info" : “Исполнитель - Название трека" } В случае ошибки API вернет следующую строку:
{"result": "error", "info" : "Информация об ошибке" }

Для успешного взаимодействия с API при обращении к странице необходимо передавать логин (login) и пароль(ра88’№огй) как POST- параметры.

login: 4707login
password:
4707pass

Приложение должно содержать активити, позволяющее просматривать внесенные в базу данных записи.

Задание 6. Работа с внешними файлами.

Разработать мобильное приложение, позволяющее пользователю асинхронно скачивать файлы журнала Научно-технический вестник. Файлы хранятся на сервере в формате PDF и расположены по адресу: http://ntv.ifmo.ru/flle/journal/идентификатор журналафйГ

Не для всех ID имеются журналы, поэтому необходимо предусмотреть сообщение об отсутствии файла. В случае если файл не найден, ответ от сервера будет содержать главную страницу сайта.

Определить существует ли файл можно по возвращаемому сервером заголовку (параметр content-type).

Примеры ссылок:
http://ntv.ifmo.ru/file/journal/1.pdf - возвращен PDF файл
http://ntv.ifmo.ru/file/journal/2.pdf - файл не найден, возвращена главная страница сайта

Файлы должны храниться на устройстве в папке, создаваемой при первом запуске приложения (путь до папки и ее название определите самостоятельно).

После окончания загрузки файла должна становиться доступной кнопка «Смотреть» и кнопка «Удалить».

При нажатии на кнопку «Смотреть» должно происходить открытие сохраненного на устройстве файла. Предусмотреть ошибку, если на устройстве не установлено приложение, открывающее PDF файлы. При нажатии на кнопку «Удалить» загруженный файл должен удаляться с устройства.

Практическая работа № 3.10. Передача данных между модулями. Тестирование мобильного приложения. Оптимизация мобильного приложения пошаговый алгоритм тестирования мобильных приложений

Цель работы: Провести тестирование созданного приложения и выполнить его оптимизацию Обеспечение качества (QA, от английского - Quality Assurance) является неотъемлемой частью

жизненного цикла разработки любых приложений, включая мобильные. К сожалению, многие упускают из виду критические особенности тестирования мобильных приложений, которые часто приводят к сбоям, ошибкам в работе приложения и плохому качеству обслуживания клиентов.

Чтобы обеспечить успешную разработку любого приложения, специалист- тестировщик должен принимать участие во всех этапах разработки - от создания концепции и анализа требований, до создания спецификаций тестирования и выпуска готового продукта. Обеспечение качества также является ключевым элементом в последующих, после прохождения этапов разработки, обзорах программного продукта.

Однако часто бывает сложно определить, с чего начать организацию процесса тестирования мобильного приложения. Для беспроблемного тестирования мы рекомендуем просто выполнить девять указанных ниже шагов.

Давайте рассмотрим особенности тестирования мобильных приложений. Цикл жизни спринтов

Этап 1: Планирование

Когда этап разработки приложения почти завершен, вы должны снова поставить перед собой вопрос - чего вы пытаетесь достичь разработкой данного приложения и какие у вас есть ограничения.

Вы должны определить следующее:

Взаимодействует ли ваше приложение с другими приложениями?

Насколько функциональны все возможности приложения?

Является ли тестируемое мобильное приложение нативным, Mobile- web или гибридным?

Ограничена ли задача тестирования приложения тестированием только внешнего интерфейса?

Стоят ли задачи на тестирование бэкенда?

Какова должна быть совместимость с различными беспроводными сетями?

Как сильно данные приложения и свободное пространство, занимаемое им, зависят от особенностей использования приложения?

Насколько быстро загружается ваше приложение, насколько быстро происходит серфинг по меню приложения и его функциям?

Как будет обрабатываться возможное увеличение нагрузки на приложение?

Влияют ли различные изменения в статусе и состоянии телефона на работу мобильного приложения?

Убедитесь, что вы договорились с командой тестировщиков о роли каждого из них и о ваших ожиданиях от процесса тестирования. В конце концов, общение является ключом к поддержанию правильной рабочей среды в команде.

Правильное понимание ролей и задач также относится и к моменту прописывания списка тест кейсов. Вся команда QA должна поддерживать и обновлять этот документ с отчетами по тестированию всех функций, реализованных на протяжении всего процесса разработки.

Этап 2. Определение необходимых типов тестирования мобильных приложений

Перед тестированием любых мобильных приложений определите, что именно в данном мобильном приложении вы хотите протестировать: набор функциональности, удобство использования, совместимость, производительность, безопасность и т. д. На этом же этапе имеет смысл выбрать методы тестирования мобильного приложения.

Определите, на какие целевые устройства направлено данное приложение, и какие требования к функционалу следует проверить.

Вы также должны определить, какие целевые устройства нужно включить в список тестирования.

Вы можете сделать это следующим образом:

- Выяснить, какие устройства будет поддерживать приложение;
- Определить, какая версия операционной системы будет самой ранней из тех, что поддерживаются приложением;
- Выявить наиболее популярные модели мобильных устройств у целевой аудитории;
- Определить набор не основных (дополнительных) устройств с экранами разных
- размеров, потенциально поддерживаемых приложением;
- Решить, будете ли вы использовать для тестирования физические устройства или их эмуляторы.

Этап 3: Тестовые случаи и разработка сценариев тестирования приложения

Подготовьте документ, описывающий тестовые случаи (test cases) для каждой тестируемой функции и функциональности.

Также перед началом тестирования важно определиться, какое сочетание ручного и автоматического тестирования вы будете применять.

При необходимости подготовьте отдельные наборы ручных тестовых случаев и сценариев для автоматического тестирования и адаптируйте их согласно требованиям проекта.

Этап 4: Ручное и автоматическое тестирование

Теперь пришло время для выполнения ручных и автоматизированных тестов. Ранее, на предыдущих этапах, вы уже определили, какие тесты и скрипты использовать и подготовили их. Теперь, на текущем этапе, вы выполняете запуск тестов для проверки механизмов основной функциональности, чтобы убедиться в отсутствии поломок.

Автоматизированное тестирование мобильных приложений хорошо экономит время и другие ресурсы тестировщиков.

Этап 5: Тестирование юзабилити и бета-тестирование

После того, как базовый функционал протестирован, настало время убедиться, что мобильное приложение является достаточно простым в использовании и обеспечивает удовлетворительный пользовательский опыт. На этом этапе необходимо поддерживать соответствие матрице кроссплатформенности, чтобы обеспечить охват пользователей различных платформ, достигнутый бета-тестерами.

Пример матрицы поддержки разных версий платформы iOs

После того, как приложение будет протестировано внутри компании, вы сможете выпустить бета-версию приложения на рынок.

Тестирование совместимости

Мобильные устройства различаются в зависимости от платформы, модели и версии их операционной системы. Важно выбрать такое подмножество устройств, которое будет соответствовать вашему приложению.

Тестирование пользовательского интерфейса

Пользовательский опыт является ключевым элементом, при тестировании приложения. Ведь наше приложение разрабатывается именно для конечных пользователей. Вам следует качественно проверить удобство использования приложения, навигацию по его элементам и контент. Тестируйте меню, опции, кнопки, закладки, историю, настройки и навигацию приложения.

Тестирование интерфейса

Тестирование пунктов меню, кнопок, закладок, истории, настроек и навигации по приложению. Тестирование внешних факторов

Приложения для мобильных устройств не будут единственными приложениями на устройстве пользователя. Вместе с вашим приложением будут установлены приложения от сторонних разработчиков. Возможно десятки таких приложений. Следовательно, вашему приложению придётся взаимодействовать с этими сторонними приложениями и прерывать работу различных функций устройства, таких как различные типы сетевых подключений, обращение к SD-карте, телефонные звонки и другие функции устройства.

Тестирование доступности

Мобильными устройствами могут пользоваться различные люди с ограниченными возможностями. По этой причине важно протестировать возможность работы с приложением людей с дальтонизмом, нарушениями слуха, проблемами пожилого возраста и другими возможными проблемами. Такое тестирование является важной частью общего тестирования юзабилити.

Этап 6: Тестирование производительности

Мобильные устройства предоставляют для приложений меньший объем памяти и меньшую доступную мощность процессора, чем стационарные компьютеры и ноутбуки. По этой причине в работе мобильных приложений очень важна эффективность использования предоставляемых ресурсов. Вам следует проверить работоспособность тестируемого приложения, изменив соединение с 2G, 3G на WIFI, проверить скорость отклика, потребление заряда батареи, стабильность работы и т. д.

Рекомендуется проверять приложение на предмет масштабируемости применения и наличие возможных проблем с производительностью.

В рамках этого этапа важно пройти и нагрузочное тестирование мобильного приложения. Функциональное тестирование

Функциональность приложения должна быть полностью протестирована. Особое внимание следует уделить установке, обновлениям, регистрации и входу в систему, обеспечению, работе со специфическими функциями устройства и сообщениям об ошибках.

Функциональное тестирование мобильного приложения, по большей части, может быть выполнено так же, как вы выполнили бы его для любого другого типа приложения. По этой причине мы не будем вдаваться в подробности этого типа тестирования. Однако следует указать области, которые имеют особое значение для мобильных приложений.

Имейте в виду, что функциональное тестирование должно включать в себя тестирование всех функций приложения и не должно быть излишне сосредоточено на какой-то одной функции.

Этап 7: Аттестационное тестирование и тестирование безопасности приложения Безопасность и конфиденциальность данных имеют огромное значение в наше время.

Пользователи требуют, чтобы вся их информация хранилась безопасно и конфиденциально.

Убедитесь, что тестируемое приложение надежно защищено. Выполните проверку на возможность внедрения SQL инъекций, на возможность перехвата сеансов, анализа дампов данных, анализа пакетов и SSL трафика.

Очень важно проверить безопасность хранилища конфиденциальных данных вашего мобильного приложения и его поведение в соответствии с различными схемами разрешений для устройств.

Помимо проверки безусловного шифрования имен пользователей и паролей, задайте себе следующие вопросы:

Есть ли у приложения сертификаты безопасности?

Использует ли приложение безопасные сетевые протоколы?

Существуют ли какие-либо ограничения, например количество попыток входа в систему до блокировки пользователей?

Этап 8: Тестирование устройства

Выполните тесты по тем алгоритмам, которые вы ранее прописали в тестовых случаях и сценариях тестирования на всех определенных для тестирования устройствах, в облаке и / или на физических устройствах.

Этап 9: контрольный этап и резюме

Этот этап включает в себя подробное и полное тестирование - от ранних итеративных этапов тестирования до регрессионных тестов, которые все еще могут потребоваться для стабилизации работы приложения и выявления незначительных дефектов.

На этом этапе тестирования вы можете добавить для проверки новые функции и изменить настройки на те, которых не будет в финальной версии.

После завершения тестирования приложения, дополнительные параметры и функции, добавленные для проверки на этом этапе, удаляются, и окончательная версия становится готовой для представления общественности.

Итоговый отчет о тестировании

Весь процесс тестирования мобильных приложений должен быть тщательно задокументирован. Проверьте дважды, сделаны ли нужные записи, и после этого сформируйте свой окончательный отчет о тестировании (test summary report).

Этот отчет должен включать:

- Важную информацию, выявленную в результате проведенных испытаний;
- Информацию о качестве проводимого тестирования;
- Сводную информацию о качестве тестируемого мобильного приложения;
- Статистику, полученную из отчетов об различных инцидентах;
- Информацию о видах тестирования и времени, затраченном на каждый из них.
- Следует также указать в отчете, что:
- данное мобильное приложение пригодно для использования в том
- качестве, в котором заявлено;
- соответствует всем критериям приемлемости функционала и качества работы. Задание. Провести тестирование, выполнить оптимизацию системы и составить отчет

Практическая работа № 3.11. Создание тем для упрощения работы с элементами. Применение DDMS для отладки приложения. Создание лога. Списки. Работа с ориентацией экрана, применение различных layouts. Анимация. Рисование. Меню. Кнопки. Диалоговые окна. Сообщения. Мультимедиа. Shared preferences. БД SQLite. Интернет-соединение. Content providers. Работа с картами и GPS. Виджеты. Публикация приложения.

Цель работы: Изучить основы верстки. Научиться управлять интерфейсом мобильного устройства при разработке программного приложения.

Просмотрите основные сведения о классах, которые понадобятся при разработке приложения.

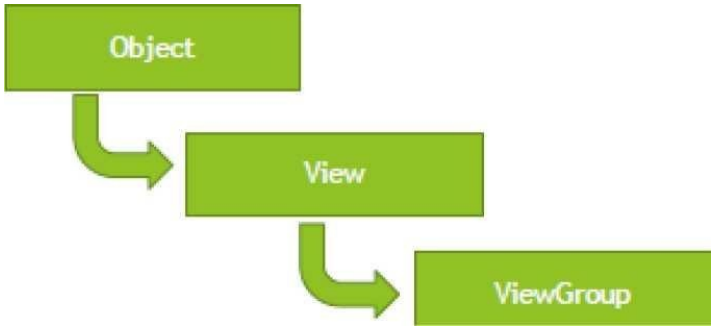


Рисунок 67 Файл разметки имеет следующую структуру

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"  android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<LinearLayout
android:layout_width="match_parent"
android:layout_height="0dp"
android:layout_weight="0.25"
android:padding="5dp">
<Button
android:layout_width="0dp"
android:layout_weight="0.33"
android:layout_height="wrap_cont
ent" android:text="Button1"
android:id="@+id/button3" android:layout_gravity="right"/>
<Button                 android:layout_width="0dp"
android:layout_weight="0.33"
android:layout_height="wrap_content"
android:text="Button2" android:id="@+id/button"/>
<Button                 android:layout_width="0dp"
android:layout_weight="0.33"
android:layout_height="wrap_content"
android:text="Button3"
android:id="@+id/button2" android:layout_gravity="center_horizontal"/>
</LinearLayout>
<LinearLayout
android:layout_width="match_parent"
android:layout_height="0dp"
android:paddingLeft="40dp"
android:paddingRight="40dp"
```

```
android:layout_weight="0.5"
android:gravity="center_vertical">
<Button
android:layout_width="0dp"
android:layout_weight="0.33"
android:layout_height="wrap_content"
android:text="Button4"
android:id="@+id/button3"/>
<Button android:layout_width="0dp"
android:layout_weight="0.33"
android:layout_height="wrap_content"
android:text="Button5" android:id="@+id/button"/>
</LinearLayout>
<LinearLayout
android:layout_width="match_parent"
android:layout_height="0dp"
android:layout_weight="0.25"
android:padding="5dp"
android:gravity="bottom">
</LinearLayout>
</LinearLayout>
```

Распространенные виды макетов

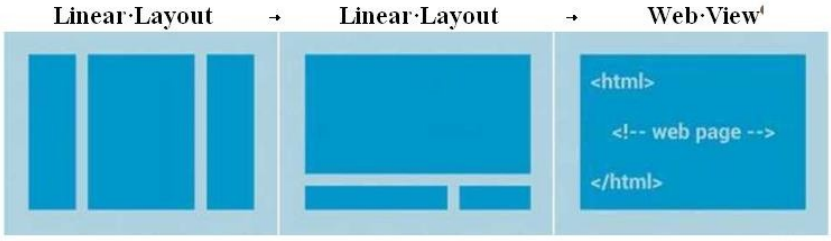


Рисунок 68

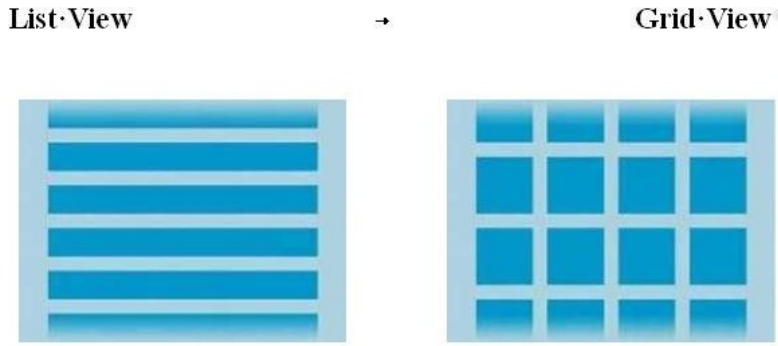


Рисунок 69

Атрибуты LinearLayout

Таблица 3

Attribute Name	Related Method	Description
android:baselineAligned	setBaselineAligned(boolean)	When set to false, prevents the layout from aligning its children's baselines.
android:baselineAligned Childindex	setBaselineAlignedChildindex(int)	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
android:divider	setDividerDrawable(Drawable)	Drawable to use as a vertical divider between buttons.
android:gravity	setGravity(int)	Specifies how an object should position its content, on both the X and Y axes, within its own bounds.
android:measureWithLargestChild	setMeasureWithLargestChildEnabled(boolean)	When set to true, all children with a weight will be considered having the minimum size of the largest child.
android:orientation	setOrientation(int)	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
android:weightSum		Defines the maximum weight sum.

Задание 1. Разработать мобильное приложение, состоящее из четырех activity.

После запуска приложения пользователь должен попадать на экран с activityt. На данном экране должно быть представлено меню, состоящее из четырех кнопок. Высота кнопок должна составлять 20% от высоты экрана. Расстояние между кнопками - 2%. Первая и последняя кнопка должны быть на равном расстоянии от краев экрана. Ширина кнопок 75%, выравнивание посередине.

После нажатия на первую кнопку пользователь должен переходить к activity2. Верстка должна осуществляться с использованием LinearLayout, ширина кнопок должна задаваться в процентах от ширины экрана.

После нажатия на вторую кнопку в activity1 пользователь должен переходить к activity3. Верстка должна осуществляться с использованием RelativeLayout (не использовать LinearLayout).

Третья кнопка в activity1 должна создавать activity3.

Кнопка должна быть выровнена по центру экрана. Цвет обводки кнопки #505050. Толщина обводки в соответствии с месяцем вашего рождения (от 1 до 12). Радиус скругления 24dp. Цвет фона экрана #FFFFFF. При нажатии на кнопку ее цвет должен изменяться на светло-зеленый.

Нажатие на четвертую кнопку в activity1 должно приводить к закрытию приложения.

Практическая работа № 3.12. Создание приложения, которое состоит из нескольких activities. Написание приложение, работающее с разными темами/стилями. Создание приложение, содержащее анимированные интерфейсные элементы

Цель работы: Знакомство с интерфейсом среды программирования. Изучение структуры проекта Рассмотрите основные понятия Android проекта.

Структура проекта

src—«исходный код» приложения (java-классы)

assets —пустая директория. Может использоваться для сохранения raw-файлов.

gen—хранилище генерируемых системных файлов. В частности, здесь располагается файл R.java, в котором хранятся идентификаторы всех ресурсов, создаваемых в проекте (строковые ресурсы и т.п.).

libs—различные библиотеки, используемые приложением
res—ресурсы проекта.
AndroidManifest.xml—файл описания проекта (поддерживаемые версии SDK, версия приложения и т.п.)
project.properties—файл, включающий настройки проекта, такие как build target. Ресурсы проекта

anim/ Содержит XML файлы, компилируемые в анимационные объекты.

color/ Содержит XML файлы описывающие цвета.
drawable/ Содержит растровые файлы (PNG, JPEG, orGIF), 9-Patch файлы, и XML файлы, описывающие Drawableshapes или Drawableobjects включающие множественные состояния(нормальное, нажатое, состояние фокуса).

layout/ Содержит XML файлы описывающие макеты экрана

menu/ Содержит XML файлы, определяющие меню приложения.

raw/ Для хранения произвольных файлов.
values/ Содержит XML файлы, компилируемые во множество видов ресурсов (strings.xmlи т.д).
xml/ СодержитXML файлы конфигурирующие компоненты приложения. Например, XML файл определяющий экран настроек.

Пример простейшего файла AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.untitled"
android:versionCode="1"
android:versionName="1.0
">
<uses-sdkandroid:minSdkVersion="19"/>
<application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
<activity android:name="MyActivity"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
```

Для начала работы требуется:

Java Development Kit
Android Software Development Kit Задание 1. Создание проекта приложения

Запустите среду программирования в IDE InelliJ Idea

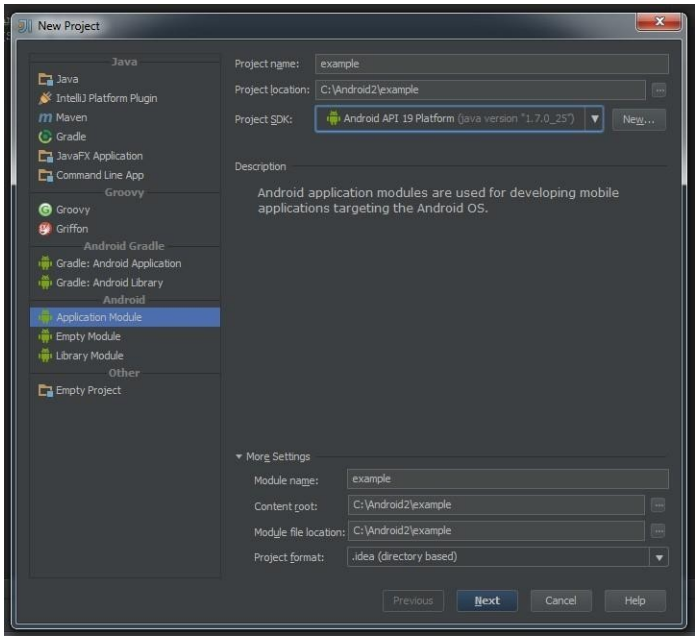


Рисунок 70

Введите данные проекта. Сохраните проект.

Задание 2. Создание приложений с одним экраном (Activity) Необходимо создать два activity и организовать переход между ними. Содержимое Activity 1 кнопка с именем btn1

Содержимое Activity 2: TextView с текстом «Переданный параметр: значение_параметра». значение_параметра – принятый параметр из Activity 1.

При запуске приложения пользователь должен попадать на экран с Activity 1. После нажатия на кнопку btn1 необходимо осуществить переход к Activity 2 и передавать параметр из Activity 1. В качестве значения передаваемого параметра использовать свою фамилию.

Ключевые классы: Activity, Intent, Button, TextView.

Практическая работа № 3.13. Создание приложения, отображающее после запуска карты Google или какие ни будь другие карты. Создание собственный виджет с настройками. Создание приложение, использующее опциональное меню (меню настроек) и контекстное меню для какого-нибудь интерфейсного элемента

Цель работы: создать приложение с картами и виджет меню

Создание андроид-приложения с картами Google Maps с использованием Google Services и Android Studio

Как зарегистрировать андроид-приложение в Google Maps API и как добавить карты Гугла в ваше приложение на android, как добавить метку на карту и как отобразить на карте текущее местоположение устройства с помощью Android Studio и службы Google Framework для создания простых картографических приложений на Android, вы узнаете в этом уроке. Нужно выполнить следующие шаги:

1. Настроить Android Studio для Maps API
2. Получить ключ API key для Google Maps Android API
3. Добавить карту на экран приложения, используя MapFragment
4. Как добавить метку на карту Google Maps
5. Как добавить текущее местоположение на карту Google

Maps Настройка Android Studio

Для того чтобы использовать новый Maps API, мы должны предоставить Google имя пакета нашего приложения. Поэтому нам необходимо создать новое приложение в Android Studio и настроить некоторые зависимости для успешного подключения к Maps API.

Откройте Android Studio и выберите создание нового проекта. На первом экране настройки нужно ввести данные, такие как имя проекта — здесь пишем MapApp, и домен компании — я укажу

адрес своего сайта fandroid.info. Имя пакета вашего приложения формируется по умолчанию из перевернутого доменного имени и имени проекта. Запомните его, оно нам еще понадобится.

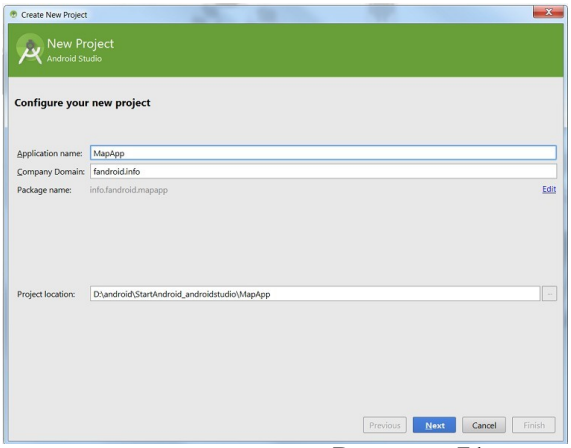


Рисунок 71

При выборе Minimum required SDK, вы можете выбрать версию ниже 4.0, однако для этого потребуется загрузка дополнительной библиотеки поддержки в Android SDK. В этом уроке мы будем устанавливать Minimum required SDK версии API 14 Android 4.0. Около 94% android-устройств будут совместимы с нашим приложением (по данным Google Play на момент публикации).

Идите вперед и жмите кнопку Далее, ничего не меняя, до конца. Если вы впервые создаете проект, используя Android Studio, среда может также скачать файлы gradle сборки. Эти файлы весят около 50мб, так что процесс создания проекта займет немного больше времени.

После создания проекта Android Studio будет индексировать его, это может занять некоторое время в зависимости от мощности вашей машины. Вы можете увидеть, что индексация закончилась, когда кнопки в верхней части экрана станут активными.

Теперь, когда проект создан, мы должны убедиться, что у нас есть необходимые компоненты SDK, для подключения нашего приложения к сервисам Google. В верхней панели инструментов, выберите кнопку SDK менеджера . Откройте папку Extras в SDK Manager и убедитесь, что у вас установлены следующие пакеты:

Еще нужно добавить в файл build.gradle нашего проекта зависимости для Google Play Services. По умолчанию есть два файла build.gradle внутри структуры проекта; один внутри папки модуля (обозначен цифрой 1 на рисунке ниже) и один в папке проекта, на верхнем уровне (с цифрой 2 на рисунке ниже).

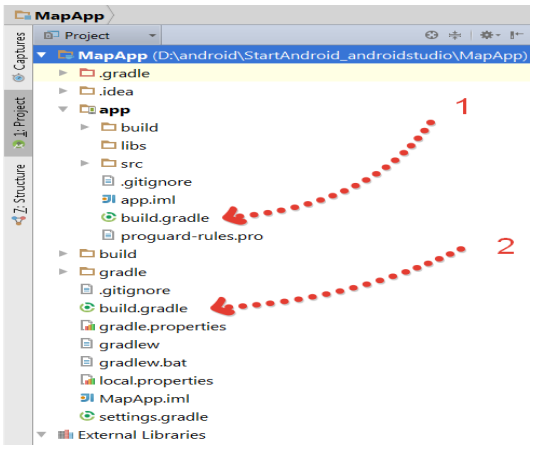


Рисунок 72

Откройте файл в папке модуля, под цифрой 1 на рисунке выше. В раздел «dependencies» нужно добавить последнюю версию библиотеки Google Play Services. По состоянию на момент написания, это

7.5.0. Вы можете узнать номер последней версии этой, а также многих других популярных библиотек, по ссылке на странице полезных ссылок, в разделе «Разное».

После добавления строки нажмите в правом верхнем углу ссылку «Sync Now» для обновления файла сборки. Обновление займет некоторое время.

Создание API ключа для приложения

Теперь мы должны зарегистрировать наше приложение в консоли разработчика Google Developer Console, чтобы позволить ему доступ к Play Services API.

Перейдите по ссылке в Google Developer Console

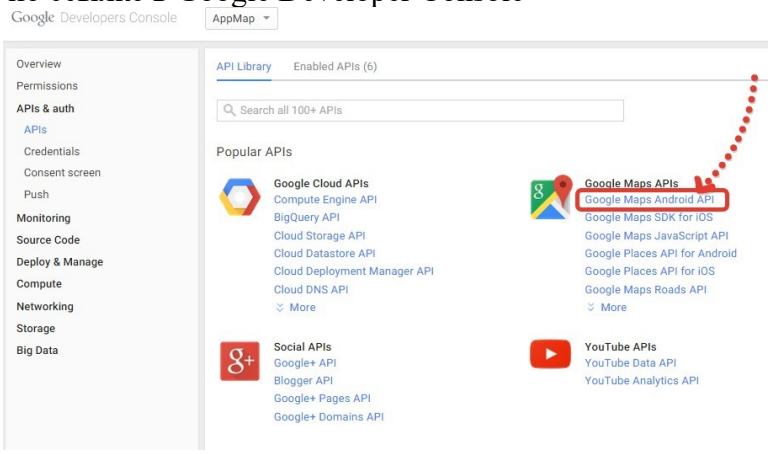


Рисунок 73

Если потребуется, создайте новый проект. После создания этого проекта, щелкните по нем и выберите “APIs & auth” в меню слева. Из списка интерфейсов выберите Google Maps Android API, и нажмите кнопку «Enable API».

После этого, выберите “Credentials” в меню слева. В разделе «Public API access» щелкните Create New Key, а затем Android Key.

Выполните следующую команду в командной строке для генерации сертификатов разработчика SHA1 fingerprint key: keytool -list -v -keystore mystore.keystore

Если вы получаете ошибку с сообщением: mystore.keystore doesn’t exist, выполните следующую команду (заменив {имя пользователя} вашим именем пользователя): keytool -list -alias androiddebugkey -keystore C:\Users\{Username}\.android\debug.keystore -storepass android -keypass android

В результате получаем SHA1 fingerprint key, который имеет вид набора пар чисел и символов, разделенных двоеточиями.

Возьмите ваш ключ SHA1, и добавьте имя пакета приложения в конец ключа, после точки с запятой. Результат будет выглядеть примерно так:

AA:95:81:D7:A7:B8:C9:79:AF:C5:7A:A6:8F:89:96:94:9A:BF:68:AB;info.fandroid.mapapp

Вставьте результат в текстовое поле и нажмите кнопку Создать. Автоматически сгенерируется API key, скопируйте его в буфер обмена.

Теперь нам нужно добавить API key в наше приложение. Переходим в Android Studio.

В проекте, откроем файл манифеста AndroidManifest.xml. Просто перед тегом </application> необходимо объявить две декларации Мета-данных . Первый Мета-тег встраивает данные версии библиотеки Google Play Services, добавленной в приложение. Второй содержит наш API ключ, который мы ранее скопировали из консоли разработчика.

```
<meta-data
   android:name="com.google.android.gms.version"
   android:value="@integer/google_play_services_version" />

<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value="AIzaSyB9oARLFPe4wRGyeuWJq0IqZ0g84TjmjVI" />
```

Теперь нам нужно указать разрешения для нашего приложения. Мы должны добавить следующие разрешения:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"
/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Наконец, чтобы карта занимала весь экран устройства, мы объявим стиль, который удаляет строку заголовка в верхней части приложения. Для этого изменим значение атрибута android:theme:

```
android:theme="@android:style/
Theme.NoTitleBar" [wpanchor id=»3″]
```

Добавление карты на экран приложения

Теперь, когда наши библиотеки, API-ключи и разрешения настроены, пришло время добавить на экран нашего приложения карту. Открываем файл макета — activity_main.xml , если вы не переименовали его во время создания проекта. Мы будем добавлять map view как фрагмент, а затем настроим его для загрузки содержимого на карту при старте приложения. Измените ваш макет следующим образом:

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <fragment
    android:id="@+id/mapVie
    w"
    android:name="com.google.android.gms.maps.MapFragment
    "                    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

В файле MainActivity.java, нам нужно инициализировать mapview. Для начала нам нужно объявить локальную переменную, которая ссылается на mapview. Мы также должны импортировать Google Maps library.

```
import com.google.android.gms.maps.*;
public class MainActivity extends
Activity {
  /** Local variables
  **/      GoogleMap
  googleMap;
```

Мы создадим закрытый метод, которая проверяет, имеется ли карта, попытается инициализировать ее, если это возможно, или в противном случае отображает ошибки пользователю. Обязательно добавьте следующие импорты в верхней части файла:

```
import com.google.android.gms.maps.*;
/**
 * Initialises the mapview
 */
private void createMapView(){
  /**
   * Catch the null pointer exception that
   * may be thrown when initialising the map
```

```
     */
    try
    {
      if(null == googleMap){
        googleMap                    =                    ((MapFragment)
            getFragmentManager().findFragmentById( R.id.mapView)).get
            Map();
        /**
         * If the map is still null after attempted initialisation,
         * show an error to the user
         */
        if(null        ==        googleMap)        {
          Toast.makeText(getApplicationContext(),
              "Error creating map",Toast.LENGTH_SHORT).show();
        }
      }
    }      catch      (NullPointerException
      exception){        Log.e("mapApp",
      exception.toString());
    }
  }
```

Далее мы создадим метод, который будет добавлять маркер на карту. Добавьте следующие операторы импорта в верхней части файла:

```
import com.google.android.gms.maps.model.LatLng;
import
com.google.android.gms.maps.model.MarkerOptions;
```

Метод добавления маркера:

```
/**
 * Adds a marker to the map
 */
private void addMarker(){

  /** Make sure that the map has been initialised
  **/ if(null != googleMap){
    googleMap.addMarker(new MarkerOptions()
            .position(new LatLng(0, 0))
            .title("Marker")
            .draggable(true)
    );
  }
}
```

Осталось теперь добавить методы createMapView и addMarker в метод onCreate: @Override

```
protected        void        onCreate(Bundle
  savedInstanceState)                    {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  createMapView();
  addMarker();
}
```

Приложение теперь при запуске загружает карту и ставит маркер прямо в центре её. Удерживая маркер, можно перетащить его по карте.

[wpanchor id=»4″]

Весь код класса для добавления метки с определенными координатами и указанием своего местоположения на карту Google maps используйте такой код:

```
package ...
import android.os.Bundle;
import
android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.Toast;
import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import
com.google.android.gms.maps.GoogleMap;
import
com.google.android.gms.maps.MapFragment;
import
com.google.android.gms.maps.model.CameraPosition;
import com.google.android.gms.maps.model.LatLng;
import
com.google.android.gms.maps.model.MarkerOptions;
public class MapActivity extends AppCompatActivity {
  GoogleMap googleMap;
  //координаты для маркера
  private static final double TARGET_LATITUDE =
  17.893366; private static final double
  TARGET_LONGITUDE = 19.511868; @Override
  protected void onCreate(Bundle
    savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_map);
    createMapView();
    addMarker();
    //добавляем на карту свое
    местоположение
    googleMap.setMyLocationEnabled(true);
  }
//создаем карту
  private void
    createMapView(){ try {
      if(null == googleMap){
        googleMap = ((MapFragment)
          getFragmentManager().findFragmentById( R.id.mapView)).get
          Map();
        if(null == googleMap) {
          Toast.makeText(getApplicationConte
          xt(),
              "Error creating map",Toast.LENGTH_SHORT).show();
        }
      }
    } catch (NullPointerException
      exception){ Log.e("mapApp",
      exception.toString());
    }
  }
  //добавляем маркер на
  карту private void
  addMarker(){
```

```
    double lat =
    TARGET_LATITUDE; double lng
    = TARGET_LONGITUDE;
  //устанавливаем позицию и масштаб отображения карты
    CameraPosition cameraPosition = new
    CameraPosition.Builder()
          .target(new LatLng(lat, lng))
          .zoom(15)
          .build();
    CameraUpdate cameraUpdate =
    CameraUpdateFactory.newCameraPosition(cameraPosition);
    googleMap.animateCamera(cameraUpdate);
    if(null != googleMap){
      googleMap.addMarker(new
      MarkerOptions()
            .position(new LatLng(lat, lng))
            .title("Mark")
            .draggable(false)
      );
    }
  }
}
```

Практическая работа № 3.14. Создание первого приложения под Android. Основы разработки интерфейсов мобильных приложений

Цель работы: разработка простого приложения, помогающего понять структуру приложения, освоить основные операторы, привыкнуть к среде разработки, основ разработки интерфейсов мобильных приложений.

Приступая к разработке мобильных приложений хорошо бы иметь представление о том, какие виды приложений существуют. Дело в том, что если удастся определить к какому типу относится приложение, то становится понятнее на какие моменты в процессе его разработки необходимо обращать основное внимание. Можно выделить следующие виды приложений:

- Приложения переднего плана выполняют свои функции только, когда видимы на экране, в противном же случае их выполнение приостанавливается. Такими приложениями являются, например, игры, текстовые редакторы, видеопроигрыватели. При разработке таких приложений необходимо очень внимательно изучить жизненный цикл активности, чтобы переключения в фоновый режим и обратно проходили гладко (бесшовно), т. е. при возвращении приложения на передний план было незаметно, что оно вообще куда-то пропадало. Для достижения этой гладкости необходимо следить за тем, чтобы при входе в фоновый режим приложение сохраняло свое состояние, а при выходе на передний план восстанавливало его. Еще один важный момент, на который обязательно надо обратить внимание при разработке приложений переднего плана, удобный и интуитивно понятный интерфейс.

- Фоновые приложения после настройки не предполагают взаимодействия с пользователем, большую часть времени находятся и работают в скрытом состоянии. Примерами таких приложений могут служить, службы экранирования звонков, SMS- автоответчики. В большинстве своем фоновые приложения нацелены на отслеживание событий, порождаемых аппаратным обеспечением, системой или другими приложениями, работают незаметно. Можно создавать совершенно невидимые сервисы, но тогда они будут неуправляемыми. Минимум действий, которые необходимо позволить пользователю: санкционирование запуска сервиса, настройка, приостановка и прерывание его работы при необходимости.

- Смешанные приложения большую часть времени работают в фоновом режиме, однако допускают взаимодействие с пользователем и после настройки. Обычно взаимодействие с пользователем сводится к уведомлению о каких-либо событиях. Примерами таких приложений могут служить мультимедиа-проигрыватели, программы для обмена тексто-

выми сообщениями (чаты), почтовые клиенты. Возможность реагировать на пользовательский ввод и при этом не терять работоспособности в фоновом режиме является характерной особенностью смешанных приложений. Такие приложения обычно содержат как видимые активности, так и скрытые (фоновые) сервисы, и при взаимодействии с пользователем должны учитывать свое текущее состояние. Возможно потребуется обновлять графический интерфейс, если приложение находится на переднем плане, или же посылать пользователю уведомления из фонового режима, чтобы держать его в курсе происходящего. И эти особенности необходимо учитывать при разработке подобных приложений.

- Виджеты - небольшие приложения, отображаемые в виде графического объекта на рабочем столе. Примерами могут служить, приложения для отображения динамической информации, такой как заряд батареи, прогноз погоды, дата и время. Разумеется, сложные приложения могут содержать элементы каждого из рассмотренных видов. Планируя разработку приложения, необходимо определить способ его использования, только после этого приступать к проектированию и непосредственно разработке.

Обратим внимание на организацию исполнения приложений в ОС Android. Как уже было отмечено приложения под Android разрабатываются на языке программирования Java, компилируется в файл с расширением .apk, после этот файл используется для установки приложения на устройства, работающие под управлением Android. После установки каждое Android приложение "живет" в своей собственной безопасной "песочнице", рассмотрим, как это выглядит:

- операционная система Android является многопользовательской ОС, в которой каждое приложение рассматривается как отдельный пользователь;
- по умолчанию, система назначает каждому приложению уникальный пользовательский ID, который используется только системой и неизвестен приложению;
- система устанавливает права доступа ко всем файлам приложения следующим образом: доступ к элементам приложения имеет только пользователь с соответствующим ID;
- каждому приложению соответствует отдельный Linux процесс, который запускается, как только это необходимо хотя бы одному компоненту приложения, процесс прекращает работу, когда ни один компонент приложения не использует его или же системе требуется освободить память для других (возможно, более важных) приложений;
- каждому процессу соответствует отдельный экземпляр виртуальной машины Dalvik, в связи с этим код приложения исполняется изолировано от других приложений.

Перечисленные идеи функционирования приложения в ОС Android реализуют принцип минимальных привилегий, т. е. каждому приложению, по умолчанию, разрешен доступ только к компонентам, необходимым для его работы и никаким больше. Таким образом обеспечивается очень безопасная среда функционирования приложений.

Однако, в случае необходимости приложения могут получить доступ к данным других приложений и системным сервисам (услугам). В случае, когда двум приложениям необходимо иметь доступ к файлам друг друга, им присваивается один и тот же пользовательский ID. Для экономии системных ресурсов такие приложения запускаются в одном Linux процессе и делят между собой один и тот же экземпляр виртуальной машины, в этом случае приложения также должны быть подписаны одним сертификатом. В случае же, когда приложению требуется доступ к системным данным, например, контактам, SMS сообщениям, картам памяти, камере, Bluetooth и т. д., пользователю необходимо дать приложению такие полномочия во время установки его на устройство.

Визуальный дизайн интерфейсов.

Силы, вложенные в разработку модели поведения программного продукта, будут потрачены впустую, если вы не сумеете должным образом донести до пользователей принципы этого поведения. В случае мобильных продуктов это делается визуальными средствами - путем отображения объектов на дисплее (в некоторых случаях целесообразно использовать тактильные ощущения от нажатия).

Визуальный дизайн интерфейсов - очень нужная и уникальная дисциплина, которую следует применять в сочетании с проектированием взаимодействия и промышленным дизайном. Она способна серьезно повлиять на эффективность и привлекательность продукта, но для полной реализации этого потенциала нужно не откладывать визуальный дизайн на потом, а сделать его одним из основных инструментов удовлетворения потребностей пользователей и бизнеса.

Изобразительное искусство, визуальный дизайн интерфейсов и прочие дисциплины дизайна Художники и визуальные дизайнеры работают с одними и теми же изобразительными

средствами, однако их деятельность служит различным целям. Цель художника - создать объект, взгляд на который вызывает эстетический отклик. Изобразительное искусство - способ самовыражения художника. Художник не связан почти никакими ограничениями. Чем необычнее и своеобразнее продукт его усилий, тем выше он ценится.

Дизайнеры создают объекты, которыми будут пользоваться другие люди. Если говорить о дизайнерах визуальных интерфейсов, то они ищут наилучшее представление, доносящее информацию о поведении программы, в проектировании которой они принимают участие. Придерживаясь целеориентированного подхода, они должны стремиться представлять поведение и информацию в понятном и полезном виде, который поддерживает маркетинговые цели организации и эмоциональные цели персонажей. Разумеется, визуальный дизайн пользовательских интерфейсов не исключает эстетических соображений, но такие соображения не должны выходить за рамки функционального каркаса.

Графический дизайн и пользовательские интерфейсы

Графические дизайнеры обычно очень хорошо разбираются в визуальных аспектах и хуже представляют себе понятия, лежащие в основе поведения программного продукта и взаимодействия с ним. Они способны создавать красивую и адекватную внешность интерфейсов, а кроме того - привносить фирменный стиль во внешний вид и поведение программного продукта. Для таких специалистов дизайн или проектирование интерфейса есть в первую очередь тон, стиль, композиция, которые являются атрибутами бренда, во вторую очередь - прозрачность и понятность информации и лишь затем - передача информации о поведении посредством ожидаемого назначения.

Дизайнерам визуальной части интерфейса необходимы некоторые навыки, которые присущи графическим дизайнерам, но они должны еще обладать глубоким пониманием и правильным восприятием роли поведения. Их усилия в значительной степени сосредоточены на организационных аспектах проектирования. В центре их внимания находится соответствие между визуальной структурой интерфейса с одной стороны и логической структурой пользовательской ментальной модели и поведения программы - с другой. Кроме того, их заботит вопрос о том, как сообщать пользователю о состояниях программы и что делать с когнитивными аспектами пользовательского восприятия функций.

Визуальный информационный дизайн

Информационные дизайнеры работают над визуализацией данных, содержимого и средств навигации. Усилия информационного дизайнера направлены на то, чтобы представить данные в форме, способствующей их верному истолкованию. Результат достигается через управление визуальной иерархией при помощи таких средств, как цвет, форма, расположение и масштаб. Распространенными объектами информационного дизайна являются всевозможные графики, диаграммы и прочие способы отображения количественной информации.

Чтобы создавать привлекательные и удобные пользовательские интерфейсы, дизайнер интерфейсов должен владеть базовыми визуальными навыками - пониманием цвета, типографики, формы и композиции - и знать, как их можно эффективно применять для передачи поведения и представления информации, для создания настроения и стимулирования физиологических реакций. Дизайнеру интерфейса также требуется глубокое понимание принципов взаимодействия и идиом интерфейса, определяющих поведение продукта.

Строительные блоки визуального дизайна интерфейсов

Дизайн интерфейсов сводится к вопросу о том, как оформить и расположить визуальные элементы таким образом, чтобы внятно отразить поведение и представить информацию. Каждый элемент визуальной композиции имеет ряд свойств, и сочетание этих свойств придает элементу смысл. Пользователь получает возможность разобраться в интерфейсе благодаря различным способам приложения этих свойств к каждому из элементов интерфейса. В тех случаях, когда два объекта обладают общими свойствами, пользователь предположит, что эти объекты связаны или похожи. Когда пользователи видят, что свойства отличаются, они предполагают, что объекты не связаны.

Создавая пользовательский интерфейс, проанализируйте перечисленные ниже визуальные свойства каждого элемента или группы элементов. Чтобы создать полезный и привлекательный пользовательский интерфейс, следует тщательно поработать с каждым из этих свойств.

Форма

Форма - главный признак сущности объекта для человека. Мы узнаем объекты по контурам. Если мы увидим на картинке синий ананас, мы его сразу опознаем, потому что мы помним его форму. И лишь потом мы удивимся странному цвету. При этом различение форм требует большей концентрация внимания, чем анализ цвета или размера. Поэтому форма - не лучшее свойство для создания контраста, если требуется привлечь внимание пользователя.

Размер

Более крупные элементы привлекают больше внимания, особенно если они значительно превосходят размерами окружающие элементы. Люди автоматически упорядочивают объекты по размеру и склонны оценивать их по размеру; если у нас есть текст в четырех размерах, предполагается, что относительная важность текста растет вместе с размером и что полужирный текст более важен, чем текст с нормальным начертанием. Таким образом, размер - полезное свойство для обозначения информационных иерархий.

Цвет

Цветовые различия быстро привлекают внимание. В некоторых профессиональных областях цвета имеют конкретные значения, и этим можно пользоваться. Так, для бухгалтера красный цвет - отрицательные результаты, а черный - положительные.

Цвета приобретают смыслы и благодаря социальным контекстам, в которых проходит наше взросление. Например, белый цвет на Западе ассоциируется с чистотой и миром, а в Азии и арабских странах - с похоронами и смертью. При этом цвет изначально не обладает свойством упорядоченности и не выражается количественно, поэтому далеко не идеален для передачи информации такого рода. Кроме того, не следует делать цвет единственным способом передачи информации, поскольку цветовая слепота встречается довольно часто.

Применяйте цвет с умом. Чтобы создать эффективную визуальную систему, позволяющую пользователю выявлять сходства и различия объектов, используйте ограниченный набор цветов - эффект радуги перегружает восприятие пользователя и ограничивает возможности по передаче ему информации.

Выбор цветовой палитры для программы необходимо проводить очень осторожно. По разным данным, той или иной формой цветовой слепоты страдают до 10% мужчин, и использование, например, красного и зеленого цветов для указания контраста затрудняет работу с приложением для этих людей.

Яркость

Понятия темного и светлого обретают смысл преимущественно в контексте яркости фона. На темном фоне темный текст почти не виден, тогда как на светлом он будет резко выделяться. Контрастность люди воспринимают легко и быстро, так что значение яркости может стать хорошим инструментом привлечения внимания к тем элементам, которые требуется подчеркнуть. Значение яркости - также упорядоченная переменная, например, более темные (с более низкой яркостью) цвета на карте легко интерпретируются: они обозначают большие глубины или большие значения других параметров.

Направление

Направление полезно, когда требуется передавать информацию об ориентации (вверх или вниз, вперед или назад). Помните, что восприятие направления может быть затруднено в случае некоторых форм и при малых размерах объектов, поэтому ее лучше использовать в качестве вторичного признака. Так, если требуется показать, что рынок акций пошел вниз, можно использовать направленную вниз стрелку красного цвета.

Текстура

Разумеется, изображенные на экране элементы не обладают настоящей текстурой, но способны создавать ее видимость. Текстура редко бывает полезна для передачи различий или привлечения внимания, поскольку требует значительной концентрации на деталях. И тем не менее текстура может быть важной подсказкой. Засечки и выпуклости на элементах пользовательского интерфейса обычно

указывают, что элемент можно перетаскивать, а фаски или тени у кнопки усиливают ощущение, что ее можно нажать.

Расположение

Расположение - это переменная, упорядоченная и выражаемая количественно, а значит, полезная для передачи иерархии. Расположение также может служить средством создания пространственных отношений между объектами на экране и объектами реального мира (например, небо в верхней половине, земля в нижней).

Задания:

1. Изучить рекомендуемую литературу.
2. Создать новое приложение и изучить его структуру.
3. Настроить интерфейс приложения;
4. Реализовать логику приложения.
5. Изучить элементы интерфейса.
6. Практическим путём научиться размещать элементы и менять их свойства.
7. Разработать прототип интерфейса собственного приложения.
8. Ответить на контрольные вопросы.
9. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Каково устройство платформы Android?
2. Что представляет собой Android SDK?
3. Назовите основные средства разработки под Android.
4. Перечислите достоинства и недостатки эмуляторов Android.
5. Выясните объем продаж мобильных устройств с ОС Android.
6. Какая версия платформы наиболее популярна в настоящее время?
7. Особенности визуального дизайна интерфейсов, строительных блоках и элементах управления.
8. Особенности проектирования GUI под Android.
9. Принципы разработки удобных пользовательских интерфейсов для мобильных приложений.

Практическая работа № 3.15. Создание многоэкранного приложения

Цель работы: научиться создавать приложения, состоящие из нескольких активностей, и диалоговые окна, а также познакомиться с элементами тач-интерфейса.

Для мобильных приложений главным ограничением является размер экрана устройства. Очень часто невозможно разместить все элементы полнофункционального приложения так, чтобы их можно было увидеть одновременно. Очевидным решением этой проблемы является разделение интерфейса на части по какому-либо принципу. Основные пути решения этой проблемы:

- Использовать различные сообщения (диалоговые окна, уведомления, всплывающие подсказки). Этот способ наиболее прост и не требует редактирования файла манифеста, однако очевидно, что так можно решить только часть задач.

- Использовать в одном приложении несколько активностей. Способ универсальный и подходит для любых приложений, однако прежде чем его реализовывать, необходимо очень хорошо продумать структуру будущего приложения. Здесь требуется редактировать манифест и организовать переключение между различными активностями удобным для пользователя способом.

- Разместить компоненты на активности таким образом, что в нужный момент можно будет легко переключиться на работу с другой частью интерфейса.

Задания:

1. Изучить рекомендуемую литературу.
2. Подумайте над собственным приложением, сочетающим различные возможности проектирования многооконных приложений, рассмотренные выше. Создайте прототип этого приложения и настройте его пользовательский интерфейс.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Использование класса Dialog
2. Варианты отображения уведомлений.
3. Назначение всплывающих подсказок.
4. Особенности разработки приложения, содержащего несколько активностей

a. Практическая работа № 3.16. Демонстрации распознавания стандартных жестов.

Принципы работы с жестами вводимыми пользователями

Цель работы: разработать простейшие приложения для демонстрации распознавания стандартных жестов, разработка приложения, помогающего понять принципы работы с жестами вводимыми пользователями.

Известно, что смартфон является "умным телефоном": предполагает обязательное наличие операционной системы и возможность установки дополнительных приложений, существенно расширяющих функционал устройства. С одной стороны, смартфон выполняет все привычные функции мобильного телефона и, благодаря компактным размерам, всегда под рукой. С другой стороны, благодаря наличию процессора и операционной системы, позволяет выполнять многие функции полноценного компьютера. Дополнительно ко всему, смартфоны обладают рядом интересных особенностей, не характерных для телефонов и компьютеров.

Для начала обратим внимание на экран смартфона. В современных смартфонах экран занимает практически всю площадь передней панели устройства, имеет высокое разрешение и является чувствительным к прикосновениям. Благодаря такой чувствительности, для взаимодействия с устройством и его приложениями можно использовать виртуальные элементы управления, чаще всего кнопки, отображаемые на экране. В связи, с чем отпадает необходимость в физических кнопках. В смартфонах реализуется, так называемый, touch-интерфейс - интерфейс, основанный на виртуальных элементах управления, выбор которых выполняется простым касанием, а также на использовании жестов (gestures). Если точек касания несколько (т. е. используется несколько пальцев), такой интерфейс, уже называется multi-touch.

Еще одна особенность смартфонов состоит в том, что для большинства их владельцев не последнюю роль играет возможность использования этого "умного телефона" в качестве аудио или видеоплеера, поэтому современные устройства становятся все более и более мультимедийными. В первой лекции обсуждалось, что в состав платформы Android входит набор библиотек для обработки мультимедиа Media Framework, в котором реализована поддержка большинства общих медиа-форматов. В связи с чем, в приложения, разрабатываемые для смартфонов под управлением Android, можно интегрировать запись и воспроизведение аудио и видео, а также работу с изображениями.

Важной и часто используемой особенностью смартфонов является наличие камеры, которая позволяет снимать все самое интересное: от первых шагов ребенка до падения метеорита. Телефон всегда под рукой и готов к работе, в связи с этим количество фотографий и небольших видеороликов резко увеличилось, и любое интересное событие в жизни индивидуума может быть запечатлено и сохранено для потомков. С ростом возможностей получения фото и видео материалов увеличивается потребность в приложениях, способ¬ных работать с этими материалами. Платформа Android позволяет разрабатывать такие приложения, которые предоставляют пользователям возможности делать фотоснимки или записывать видео, каким-то образом обрабатывать полученные материалы и использовать их далее.

Большинство смартфонов оснащены GPS-модулем, а некоторые даже комбинированным модулем GPS/ГЛОНАСС, что позволяет использовать такое устройство в качестве инструмента для ориентирования на местности. Во многих случаях смартфон с установленным соответствующим программным обеспечением вполне может заменить GPS навигатор. В разрабатываемых приложениях иногда бывает очень полезно добавить возможность получения координат устройства и хозяина, если оба находятся в одном месте, и использовать эти координаты для каких-либо целей. Например, уже существуют приложения, которые позволяют отслеживать параметры человека (спортсмена) во время преодоления некоторых расстояний бегом, на велосипеде, на лыжах и т. д. Такое приложение работает во время тренировки (устройство должно перемещаться вместе со спортсменом), по окончанию можно получить полную статистику маршрута: точное время в пути, рас-стояние, подъемы/спуски, среднюю скорость, потраченные калории и т. д. Заметим, что большая часть информации опирается на данные, полученные со спутников GPS.

Рассмотрение особенностей смартфонов будет неполным, если оставить без внимания датчики и сенсоры, которыми оснащены большинство устройств. Эти микроустройства обеспечивают связь смартфона с окружающей средой и добавляют новые удивительные функции. С помощью датчика приближения, например, можно отключать подсветку экрана при приближении телефона к уху пользователя во время разговора, блокировать экран, чтобы не было возможности случайно нажать на отбой. Акселерометр может использоваться для смены ориентации экрана, для управления в играх, особенно симуляторах, а также в качестве шагомера. Датчик освещенности позволяет регулировать яркость экрана. Гироскоп может применяться для определения более точного позиционирования устройства в пространстве.

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете android.gesture и позволяет сохранять, загружать, создавать и рас-познавать жесты.

Задания:

1. Изучить рекомендуемую литературу.
2. Разработать приложение, в котором демонстрируется распознавание всех под-держиваемых жестов.
3. Разработать приложение, в котором демонстрируется распознавание только некото-рой части поддерживаемых жестов по выбору программиста.
4. Реализовать жестами ввод чисел в приложении "Угадайка", разработанном в лабо-раторной работе второй темы. Создать жесты "0", "1", "2", "3", "4", "5", "6", "7", "g", "9" для ввода цифр и жест "S" для остановки ввода числа. В приложение добавить распознавание этих жестов, преобразование их в число и сравнение полученного числа с загаданным.
5. Разработать простой калькулятор с жестовым вводом чисел и операций.
6. Разработать блокнотик для заметок с рукописным вводом текста.
7. Ответить на контрольные вопросы.
8. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Возможности добавления сенсорного управления в мобильные приложения под Android.
2. Процесс распознавания жеста.
3. Назначение мультимедиа библиотеки Android.
4. Использование встроенной камеры.
5. Технология создания набора жестов.
6. Способы импортировать жесты в проект.
7. Использование созданных жестов в приложении.

Практическая работа № 3.17. Многооконное приложение. Геолокационные возможности.

Использование сторонних библиотек

Цель работы: разработка многооконного приложения, предоставляющего возможности: воспроизведения аудио и видео файлов, создания и отображения фотоснимков, разработка приложения, демонстрирующего геолокационные возможности.

В современных смартфонах экран занимает практически всю площадь передней панели устройства, имеет высокое разрешение и является чувствительным к прикосновениям. Благодаря такой чувствительности, для взаимодействия с устройством и его приложениями можно использовать виртуальные элементы управления, чаще всего кнопки, отображаемые на экране. В связи, с чем отпадает необходимость в физических кнопках. В смартфонах реализуется, так называемый, touch-интерфейс - интерфейс, основанный на виртуальных элементах управления, выбор которых выполняется простым касанием, а также на использовании жестов (gestures). Если точек касания несколько (т. е. используется несколько пальцев), такой интерфейс, уже называется multi-touch.

Еще одна особенность смартфонов состоит в том, что для большинства их владельцев не последнюю роль играет возможность использования этого "умного телефона" в качестве аудио или видеоплеера, поэтому современные устройства становятся все более и более мультимедийными. В первой лекции обсуждалось, что в состав платформы Android входит набор библиотек для обработки мультимедиа Media Framework, в котором реализована поддержка большинства общих медиа-форматов. В связи с чем, в приложения, разрабатываемые для смартфонов под управлением Android, можно интегрировать запись и воспроизведение аудио и видео, а также работу с изображениями.

Важной и часто используемой особенностью смартфонов является наличие камеры, которая позволяет снимать все самое интересное: от первых шагов ребенка до падения метеорита. Телефон всегда под рукой и готов к работе, в связи с этим количество фотографий и небольших видеороликов резко увеличилось, и любое интересное событие в жизни индивидуума может быть запечатлено и сохранено для потомков. С ростом возможностей получения фото и видео материалов увеличивается потребность в приложениях, способных работать с этими материалами. Платформа Android позволяет разрабатывать такие приложения, которые предоставляют пользователям возможности делать фотоснимки или записывать видео, каким-то образом обрабатывать полученные материалы и использовать их далее.

Большинство смартфонов оснащены GPS-модулем, а некоторые даже комбинированным модулем GPS/ГЛОНАСС, что позволяет использовать такое устройство в качестве инструмента для ориентирования на местности. Во многих случаях смартфон с установленным соответствующим программным обеспечением вполне может заменить GPS навигатор. В разрабатываемых приложениях иногда бывает очень полезно добавить возможность получения координат устройства и хозяина, если оба находятся в одном месте, и использовать эти координаты для каких-либо целей. Например, уже

существуют приложения, которые позволяют отслеживать параметры человека (спортсмена) во время преодоления некоторых расстояний бегом, на велосипеде, на лыжах и т. д. Такое приложение работает во время тренировки (устройство должно перемещаться вместе со спортсменом), по окончанию можно получить полную статистику маршрута: точное время в пути, расстояние, подъемы/спуски, среднюю скорость, потраченные калории и т. д. Заметим, что большая часть информации опирается на данные, полученные со спутников GPS.

Задания:

1. Изучить рекомендуемую литературу.
2. Разработать приложение, которое предоставляет пользователю возможность выбора рода деятельности: работа с камерой для создания снимков; воспроизведение аудио и видео; просмотр изображений.
3. Разработать приложение, в котором реализовать четыре активности: главная активность, предназначена для выбора рода деятельности, содержит три кнопки, нажатие на каждую кнопку вызывает к жизни соответствующую активность; активность для работы с камерой и создания снимков; активность для воспроизведения аудио и видео; активность для просмотра изображений.
4. Разработать приложения, получающего координаты устройства и отслеживающего их изменение
5. Ответить на контрольные вопросы.
6. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Как настроить интерфейс и реализовать логику активности для работы с камерой.
2. Как настроить интерфейс и реализовать логику активности для воспроизведения аудио и видео.
3. Как настроить интерфейс и реализовать логику активности для просмотра изображений.
4. Как настроить интерфейс и реализовать логику главной активности приложения.
5. Технология разработки приложения, получающего координаты устройства и отслеживающего их изменение
6. Каким образом имитировать передачу данных о местоположении на эмулятор.

Практическая работа № 3.18. Работа с базами данных в Android. Основные приемы работы с инструментами разработки

Цель работы: разработка Android приложения, демонстрирующего возможности работы с базой данных SQLite, познакомиться со средой разработки приложений для Windows Phone 8.

SQLite - небольшая и при этом мощная система управления базами данных. Эта система создана в 2000 году, ее разработчик доктор Ричард Хипп (Dr. Richard Hipp). В настоящее время является одной из самых распространенных SQL-систем управления базами данных в мире. Можно выделить несколько причин такой популярности SQLite: она бесплатная; она маленькая, примерно 150 Кбайт; не требует установки и администрирования. Подробнее см. http://www.sqlite.org.

База данных SQLite - это обычный файл, его можно перемещать и копировать на другую систему (например, с телефона на рабочий компьютер) и она будет отлично работать.

Для того чтобы начать разработку приложений для Windows Phone, нужно установить соответствующие инструментальные средства.

Windows Phone SDK 8.0.

Данный комплект средств разработчика поддерживает создание приложений для Windows Phone 8 и Windows Phone 7.5. При этом с его помощью, при установке соответствующего обновления, о котором читайте ниже, можно создавать и приложения, рассчитанные на Windows Phone 7.8. Скачать SDK можно по данной ссылке: http://www.microsoft.com/ru-ru/download/details.aspx?id=35471. Здесь можно загрузить вебинсталлятор, в некоторых случаях, например, когда установку нужно произвести на несколько компьютеров, удобнее пользоваться ISO-образом, который можно записать на DVD-диск устанавливать с него. Ссылку на загрузку образа (http://go.microsoft.com/fwlink/?LinkID=257234&clcid=0x419) можно найти в нижней части страницы, в разделе примечаний.

SDK предъявляет следующие требования к операционной системе компьютера и аппаратному обеспечению:

Поддерживаемые операционные системы:

- Windows 8; Windows 8 Pro, клиентские 64-разрядные (x64) версии Windows 8.
- Аппаратное обеспечение:
- 6,5 ГБ свободного места на жестком диске;
- 4 ГБ ОЗУ;
- 64-разрядный (x64) процес-

сор. Эмулятор Windows Phone 8:

- выпуск Windows 8 Профессиональная или более полнофункциональный
- процессор, поддерживающий трансляцию адресов второго уровня

(SLAT) Задания:

1. Изучить рекомендуемую литературу.
2. Разработать приложение, демонстрирующее возможности работы с базой данных: создание, добавление записей, просмотр записей, удаление базы данных.
3. . Создать новый проект, запустить в эмуляторе и сохранить проект приложения.
4. Ответить на контрольные вопросы.
5. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Основы работы с базами данных, SQLite.
2. Системы анимации платформа Android.
3. Требования к 2D и 3D графике.
4. Основные принципы разработки игровых приложений для смартфонов
5. Особенности установки средств разработки для платформы Windows Phone, настройки этих средств.
6. Назначение учетной записи разработчика.
7. Разблокировка устройств, позволяющих отлаживать приложения на смартфонах, работающих под управлением Windows Phone.

Практическая работа № 3.19. Шаблоны проектов, структура проектов. Элементы управления. Разработка пользовательского элемента управления

Цель работы: научиться работать с проектами приложений для Windows Phone в Visual Studio Можно отметить, что состав доступных проектов на Visual C# и Visual Basic совпадает (за

исключением проекта Приложение модульного тестирования Windows Phone, который доступен только на C#). Это - проекты, на основе которых можно создать основную часть Windows Phone- приложений, и, в том числе - игровые приложения, использующие Direct 3D, XNA-игры, которые всё еще можно разрабатывать в расчёте на Windows Phone 7, и, хотя Windows Phone 8-устройства могут исполнять такие приложения, их создание специально для платформы Windows Phone 8 не предусмотрено.

Шаблоны проектов, предусматривающие использование Visual C++ ориентированы на разработку Direct3D-приложений, обычно это игры, и на создание высокопроизводительных компонентов среды выполнения Windows Phone.

Правильный выбор шаблона приложения позволяет ускорить процесс разработки за счёт наличия в созданном по нему проекте приложения некоего стартового набора элементов. В то же время, нельзя сказать, что, например, создавая проект на основе одного шаблона, разработчик принципиально не может реализовать в таком приложении ту же функциональность, которая

предусмотрена шаблоном другими шаблонами. В общем случае проект, созданный по некоему шаблону - это лишь стартовая точка разработки, дающая простейшее работающее приложение, которое в ходе разработки претерпевает множество изменений и дополнений, превращаясь в готовый программный продукт.

В стандартной поставке среды разработки для Windows Phone имеется набор элементов управления. Просмотреть доступные элементы можно, открыв Панель элементов нажатием на её ярлык, расположенный в левой части экрана. При необходимости Панель элементов можно закрепить в видимом состоянии, для её закрепления или переключения в режим автоматического скрытия служит значок с изображением канцелярской кнопки в заголовке её окна.

Состав Панели элементов можно расширить, для этого нужно щёлкнуть по ней правой кнопкой мыши и выбрать в появившемся контекстном меню пункт «Выбрать элементы». На рис. 74. показано окно Visual Studio с открытым окном Выбор элементов панели элементов. Если вы полагаете, что какого-то стандартного элемента управления "нет на месте", вероятнее всего, его отображение просто отключено. С помощью данного окна можно включить отображение нужного элемента. Панель элементов можно вернуть к виду по умолчанию, для этого служит команда её контекстного меню Сброс панели элементов

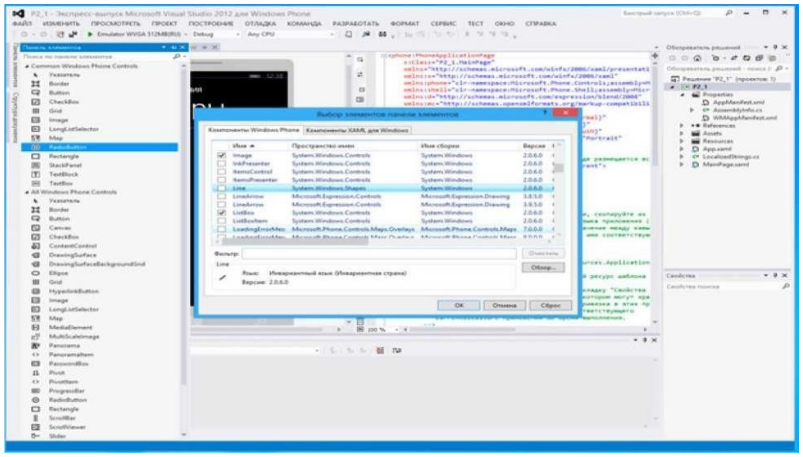


Рисунок 74

В ходе разработки приложений для Windows Phone иногда возникает необходимость в элементах управления, которых не найти ни среди стандартных элементов управления, ни среди тех, которые предлагают сторонние разработчики. Из подобной ситуации есть выход, который заключается в создании собственного элемента управления. Особенно это оправдано в тех случаях, когда подобный элемент управления нужен на нескольких страницах приложения или тогда, когда его планируется использовать в нескольких приложениях.

Задания: для выполнения лабораторной работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Создайте проект приложения по шаблону Приложение Windows Phone 8, рассчитанный на платформу Windows Phone 8, добавьте в него новую страницу и настройте проект так, чтобы именно эта страница, а не MainPage.xaml, вызывалась при запуске приложения. Используя несколько графических файлов, подходящих для формирования экрана-заставки на экранах разрешений 480x800 (WVGA),

768x1280 (WXGA) и 720x1280 (720p), добейтесь того, чтобы приложение отображало различные экраны-заставки при запуске на эмуляторах с различным разрешением экрана. Изучите раздел Возможности файла-манифеста приложения для Windows Phone, выясните назначение различных возможностей, которые можно устанавливать для приложения, пользуясь пояснениями и ссылками на документацию, которые приводятся в редакторе манифеста. Выберите 5 любых возможностей и подготовьте развёрнутое сообщение с их описанием.

3. Создайте учебное приложение, которое использует 5-8 элементов управления из пакета Coding4Fun, исследуйте их возможности и подготовьте отчёт о проделанной работе с приложением к нему копий экранов страниц приложений и описанием особенностей настройки и использования каждого из выбранных элементов управления.

4. Поищите другие свободно распространяемые пакеты элементов управления, дайте описание одного из них в отчёте.

5. Создайте проект приложения, добавьте в него пользовательский элемент управления, содержащий анимацию, которая запускается при касании этого элемента управления. Добавьте элемент управления на страницу MainPage.xaml и разместите на ней анимированную с помощью раскадровки геометрическую фигуру. Настройте поведение приложения так, чтобы раскадровка, описанная на странице MainPage.xaml запускалась после завершения раскадровки, описанной в пользовательском элементе управления

6. Ответить на контрольные вопросы.
7. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Основные особенности шаблонов проектов приложений для Windows Phone 8.
2. Особенности устройства проекта приложения, созданного по шаблону Приложение Windows Phone, рассчитанного на платформу Windows Phone OS 8.0.
3. Структура и назначение основных файлов проекта.
4. Роль и особенности файла-манифеста приложения для Windows Phone.
5. Назначение стандартных элементов управления.
6. Особенности работы со стандартными элементами управления.
7. Элементы управления сторонних разработчиков.
8. Особенности работы с пакетами элементов управления сторонних разработчиков
9. Методика создания пользовательского элемента управления.
10. Методика создания уникальных элементов управления, которые можно использовать на различных страницах приложения или в разных приложениях.

Практическая работа № 3.20. Навигация в приложении. Обмен данными внутри приложения

Цель работы: освоить методику организации навигации в приложениях, освоить методику организации обмена данными в приложениях

С точки зрения пользователя перемещения по страницам приложения происходят в ответ на взаимодействие пользователя с некими элементами управления, либо - как реакция на некоторые события, соответствующие сценарию работы приложения.

Приложение для Windows Phone содержит базовый элемент Microsoft.Phone. Controls. PhoneApplicationFrame

(http://msdn.microsoft.com/en-us/library/windowsphone/develop/microsoft. phone.controls.

phoneapplicationframe%28v=vs.105_%29.aspx), так называемую рамку, которая, в свою очередь, служит контейнером для элементов Microsoft.Phone.Controls.PhoneApplicationPage (http://msdn.microsoft.com/en-us/library/windowsphone/develop/microsoft.phone.Controls.phoneapplicationpage_%28v=vs.105_%29.aspx), то есть - для страниц приложения. Страницы, в свою очередь, могут содержать другие элементы управления, некоторые из которых служат для организации содержимого страницы, некоторые служат для отображения содержимого и организации непосредственного взаимодействия с пользователем. Элементы управления, которые отвечают за организацию содержимого страницы, это, например, System.Windows.Control.Grid, то есть - сетка для содержимого, Microsoft.Phone.Controls.Panorama - то есть элемент управления для организации панорамного просмотра данных, с общим заголовком, располагающийся на странице и содержащий элементы Microsoft.Phone.Controls.PanoramaItem. Это - элемент управления Microsoft.Phone.Controls.Pivot, который содержит элементы Microsoft.Phone.Controls.PivotItem и позволяет организовывать данные на странице, снабжая их отдельными заголовками.

Корневая рамка приложения, PhoneApplicationFrame, определяется в файле App.xaml.cs, она носит имя RootFrame, в приложении может быть только одна рамка. В свою очередь, XAML-страницы приложения - это объекты PhoneApplicationPage - их в приложении может быть столько, сколько нужно для решения задач, возложенных на него разработчиком.

Помимо обычных элементов управления, на странице можно описать так называемую панель приложения, Microsoft.Phone.Shell.ApplicationBar (http://msdn.microsoft.com/en-us/library/windowsphone/develop/microsoft.phone.shell.applicationbar%28v=vs.105%29.aspx).

Уже из пространства имён панели приложения ясно, что перед нами системный объект. Поэтому и работа с ним из кода выглядит не так, как работа с обычными элементами управления. Панель приложения располагается в нижней части страницы. Фактически, это - комбинация панели инструментов и меню. "Панель инструментов" может содержать до четырех кнопок с пиктограммами для организации быстрого доступа к наиболее востребованным командам, а в меню, при необходимости, выносятся другие команды. Если на странице нужна функциональность, подобная функциональности панели инструментов и меню, желательно реализовать её именно с помощью панели приложения. Пользователи платформы Windows Phone привыкли к подобному средству взаимодействия с приложением, знают, чего от него можно ожидать, в большинстве случаев панель приложения - это наилучший выбор. Однако, не стоит забывать о том, что если в приложении, например, в графическом редакторе, нужна особая функциональность меню и панелей инструментов, не запрещено реализовывать её средствами, избранными разработчиком. Помимо команд, которые отвечают за выполнение некоторых действий, имеющих отношение к текущей странице приложения, панель приложения можно использовать и для организации перехода на другие страницы, то есть - она является одним из инструментов, который применим в навигации по страницам.

Разрабатывая систему навигации для приложения, стоит помнить о роли кнопки Назад, которая позволяет пользователю вернуться на предыдущую страницу. Не нужно создавать элементы управления, дублирующие её функциональность. Кроме того, стоит постоянно помнить о том, что приложения для Windows Phone - это приложения для мобильных устройств. Чем проще и понятнее будет система навигации по приложению, чем меньше действий пользователю придётся совершить для того, чтобы выполнить то, ради чего он запустил приложение, тем выше шансы приложения на успех. Если без некоторого экрана в вашем приложении можно обойтись - значит избавьтесь от него, сосредоточьтесь на основной задаче приложения, помните о том, что приложением могут пользоваться в условиях, когда на то, чтобы разбираться в сложной структуре его страниц, просто нет времени.

Приложения для Windows Phone, как и любые другие приложения, нуждаются во внутренних механизмах обмена данными. То есть, в простейшем виде, пользователь может ввести некоторые данные на странице №1, эти данные могут понадобиться на странице №2. При этом возможен как сценарий, когда со страницы №1 осуществляется прямой переход на страницу №2, а возможно, что "страница №1" - это страница настроек, данные, введенные на которой должны быть доступны всем остальным страницам. Более того, если расширить эту идею, обмен данными внутри приложения - это не только обмен информацией между страницами, и не только обмен информацией, которую вводит пользователь. Например, в приложении может быть описан некий объект, не имеющий визуального

представления, но работающий с данными, которые должны поступать в него из других частей приложения. Приложение может, например, загружать некоторые данные из Интернета, или, скажем, получать ключ доступа к учетной записи пользователя после успешной авторизации в интернет-сервисе. Подобные данные или ключи могут никогда не выводиться в интерфейсе, но они могут быть нужны для обеспечения работоспособности различных механизмов.

Задания:

1. Изучить рекомендуемую литературу.
2. Продумайте систему навигации по приложению, идею которого вы выработали после проведения предыдущего семинарского занятия. Подумайте над тем, нужна ли в вашем приложении панель приложения и если нужна - определите примерный состав её кнопок и команд. Создайте новый проект, добавьте в него страницы, которые будут содержаться в вашем приложении, при необходимости - настройте панели приложения и организуйте навигацию между ними в соответствии с особенностями приложения. Убедитесь в том, что навигация по приложению интуитивно понятна, предложите кому-нибудь поработать с системой навигации и оценить - не вызывает ли работа с ней затруднений.
3. Продумайте систему обмена данными между страницами приложения, над котором вы работаете. Подумайте, где достаточно будет передачи данных с использованием параметров, передаваемых странице при её вызове, а где понадобится использовать общедоступное поле, объявленное в классе App. В учебных целях реализуйте оба механизма.
4. Ответить на контрольные вопросы.
5. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Технология организации навигации по приложению с использованием кнопок, кнопок- гиперссылок, элементов управления панели приложения.
2. Технология организации обработки событий переходов страниц.
3. Технология управления стеком переходов для организации особых сценариев навигации.
4. Механизмы, которые можно использовать при передаче данных внутри приложения.
5. Передача параметров при вызове страницы и использование общедоступного поля класса App.
6. Построение строки параметров вызывающей страницей и обработка этих параметров вызываемой страницей.

Практическая работа № 3.21. Использование шаблона проектирования MVVM. Работа с JSON, XML, сжатие данных

Цель работы: освоить методику создания приложений с использованием MVVM, освоить технологию работы с форматами JSON и XML и работу со сжатием данных.

Простейшее приложение для Windows Phone, всеми аспектами которого занимается один разработчик, вполне можно написать, не задумываясь о том, чтобы разделять код, имеющий отношение к визуализации данных, к их обработке и получению из некоторых источников. Приложение, в котором большая часть кода, ответственного за всю его функциональность, собрана в программном файле к его начальной странице (особенно, когда эта функциональность требует пары десятков строк кода), будет работать. Но небольшой рост сложности приложения приводит к тому, что таким кодом становится очень неудобно управлять. Его неудобно расширять, тестировать, неудобно привлекать других программистов к работе над приложением. Небольшое изменение в коде может повлечь за собой

множество неожиданных, неприятных последствий. Идея разделения кода, который ответственен за работоспособность различных подсистем приложения, это одна из идей, которая легла в основу создания шаблонов проектирования приложений. Один из таких шаблонов, получивший популярность в последние годы, называется MVVM, Model-ViewViewModel, Модель-Представление- Модель представления. Этот шаблон используют на разных платформах, он универсален и позволяет создавать крупные программные проекты, обладающие чёткой и понятной структурой. Для того чтобы приступить к использованию MVVM, нужно понять сущность некоторых основных понятий, которые имеют отношение к этому шаблону проектирования.

Очень кратко сущность MVVM можно выразить так: "Модель содержит исходные данные, которые готовятся к выводу с помощью модели представления и отображаются пользователю с помощью представления".

Данные, которые передаются между веб-сервисом и клиентским приложением, кодируют с использованием определенных форматов. Существует множество таких форматов. Например, в ответ на запрос загрузки веб-страницы с веб-сервера могут поступить данные в формате HTML. При работе с API веб-сервисов обычно используются другие форматы, среди них наиболее распространены JSON (Java Script Object Notation) и XML (eXtensible Markup Language). Эти форматы могут использоваться и в задачах, которые не связаны с веб-службами. Например, в XML или JSON могут быть сериализованы объекты (при условии возможности сериализации), состояние которых нужно сохранить в постоянной памяти и загрузить, десериализовав, при очередном запуске приложения.

Как мы увидим ниже, JSON и XML имеют определенные особенности, которые можно учитывать, принимая решение об использовании того или иного формата. С их помощью можно закодировать одни и те же данные, существенная разница между ними будет заключаться в размере полученных данных и в удобстве восприятия данных человеком (нужно обычно при отладке). Ниже мы рассмотрим эти характеристики на практике.

С вопросами передачи и хранения данных тесно связана тема сжатия данных. Сжатие позволяет особым образом обработать данные, получив их представление, имеющее меньший размер, чем исходные данные, но содержащее ту же информацию. В данном случае речь идёт о так называемом сжатии информации без потерь, такое сжатие применимо, например, к документам (или к любым другим данным), когда критически важно точное соответствие данных, хранящихся в исходном документе тем данным, которые получены из сжатой копии документа. Среди алгоритмов сжатия без потерь можно отметить, например, алгоритм .ZIP.

Существует и так называемое сжатие с потерями - оно используется в тех случаях, когда отбрасывание некоторой части информации не ухудшает (или ухудшает незначительно) возможности по работе с информацией. Взамен потери некоторой части информации мы получаем очень большие уровни сжатия. Такое сжатие используется для кодирования изображений (.JPG), звуковых файлов (.MP3), видеофайлов.

Задания:

1. Изучить рекомендуемую литературу.
2. Проработать пример, приведенный в лабораторной работе, чтобы лучше понять работу механизмов приложения, построенного по шаблону MVVM.
3. Самостоятельно ознакомиться со следующими примерами: "Windows Phone Starter Kit for RSS - WP8" ("Стартовый проект для разработки RSS-приложений для Windows Phone 8", http://code.msdn.microsoft.com/Windows-Phone-Starter-Kit- 390ee0ef ; "Sharing Code between Windows Store and Windows Phone App (PCL + MVVM + OData)" ("Совместное использование кода в приложениях для Магазина Windows и для Windows Phone (PCL + MVVM + OData)"), http://code.msdn.microsoft.com/Sharing-Code-between-411c999b
4. Подумайте, как ваше приложение может воспользоваться возможностями сериализации и десериализации в организации его работы. Оцените возможную выгоду от использования возможностей сжатия данных при организации хранения данных вашего приложения. На основе примера, приведенного в лабораторной работе, создайте приложение, которое позволяет оценить скорость сериализации объекта, объем данных которого можно регулировать в диапазоне 10 - 50 Мб, и десериализации данных в такой объект. Используйте в эксперименте

форматы JSON и XML Для оценки времени можно воспользоваться возможностями получения текущего времени в начале и в конце операции

5. Ответить на контрольные вопросы.
6. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Методика реализации приложения, использующего шаблон MVVM.
2. Работа механизмов приложения, построенного по шаблону MVVM
3. Работа с JSON, XML.
4. Технология сжатия данных.
5. Выбор формата передачи данных позволяющий создать приложение, эффективно использующее системные ресурсы.
6. Затраты вычислительных ресурсов при сжатии данных.

Практическая работа № 3.22. Работа webClient и HttpWebRequest

Цель работы: освоить технологию работы с классами webClient и HttpWebRequest.

Для работы с веб-сервисами из приложений для Windows Phone обычно используют классы System.Net. WebClient (http://msdn.microsoft.com/ en- us/library/ windowsphone/develop/ system.net.webclient%28v =vs.105%29.aspx) и System.Net.HttpWebRequest http:// msdn.microsoft.com/en-us/library/windowsphone/develop/system.net.httpwebrequest%28v=vs.105%29. aspx). И тот и другой класс позволяют достигать схожих целей, однако, между ними есть некоторые отличия, которые и определяют выбор того или иного класса для использования в конкретном приложении.

Так, обычно WebClient применяют тогда, когда нужно получить какие-либо данные из Интернета, он обеспечивает простую и удобную работу с GET/POST запросами. То есть, если наша задача - это получение, например, RSS-ленты с сервиса, или отправка файла с использованием POST- запроса, возможностей WebClient для этого вполне хватит. Кроме того, его использование позволит

упростить код. Его возможностей хватит и для выполнения многих других действий, но более сложные сценарии взаимодействия с вебслужбами обычно реализуют с использованием HttpWebRequest. Этот класс, в частности, нужен там, где предполагается использование PUT/DELETE запросов, он предоставляет больший уровень контроля над параметрами запроса. Например, если речь идёт об отправке файла на веб-сервер, то обычно для этого нам понадобится HttpWebRequest. При работе над конкретным проектом стоит ознакомиться с наборами инструментов, доступных в пространстве имен System.Net

(http://msdn.microsoft.com/en-
us/library/windowsphone/develop/btdf6a7e%28v=vs.105%29.aspx).

Можно сказать, в итоге, что для выполнения простых задач использование WebClient позволяет упростить их решение, при прочих равных условиях HttpWebRequest потребует более сложных программных конструкций, больше настроек. А при выполнении задач более сложных, требующих более полного контроля над процессом взаимодействия с веб-службой, некоторая усложнённость использования HttpWebRequest вполне оправдана.

Задания:

1. Изучить рекомендуемую литературу.
2. Если приложение, над которым вы работаете, подразумевает работу с каким-либо веб- сервисом, получение данных из Интернета, подумайте над тем, какие из рассмотренных механизмов вы сможете в них использовать. В частности, исходя из

круга задач, которые ваше приложение будет решать с использованием интернет-сервисов аргументируйте использование в нём таких средств, как классы HttpWebRequest или WebClient, задача вызова веб-браузера из приложения, элемент управления, который позволяет встраивать веб-браузер в страницу приложения.

3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Назначение HttpWebRequest и WebClient.
2. Методы асинхронного выполнения задач.
3. Средства выполнения асинхронных вызовов.

Практическая работа № 3.23. Работа с API веб-сервисов

Цель работы: освоить технологию работы с API веб-сервисов.

Если разработчик хочет создать приложение, взаимодействующее с неким вебсервисом, сначала следует изучить интерфейс программирования (API), который реализован на данном сервисе и позволяет организовывать взаимодействие с ним приложений. Подавляющее большинство сервисов поддерживают те или иные API, иногда речь идёт о нескольких интерфейсах, построенных с использованием различных технологий. На начальном этапе подготовки к разработке следует выяснить особенности их использования. Обычно владельцы сервисов положительно относятся к разработчикам приложений, так как такие приложения, фактически, расширяют аудиторию сервиса, расширяют его присутствие на различных платформах. В данном случае речь идёт о платформе Windows Phone.

Поэтому на веб-сайтах сервисов обычно имеется документация по их API, которая содержит много полезной информации. Для того, чтобы найти такие разделы, обычно нужно приложить некоторые усилия для их поисков, например, выполнить поисковый запрос по названию сервиса с добавлением ключевых слов "API" или "Development". Например, поиск по ключевым словам "twitter API" позволяет найти ссылку на портал разработчиков Twitter: https://dev.twitter.com/, рис. 75. где в разделе Documentation (Документация) можно найти интересующие нас материалы. Изучение справочного раздела сервиса Twitter позволяет узнать о том, что этот сервис поддерживает REST-API, для авторизации в сервисе используется протокол OAuth.

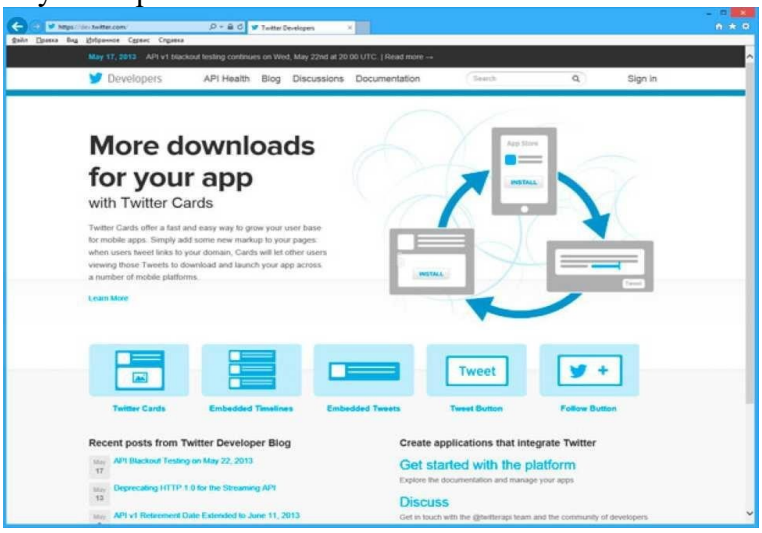


Рисунок 75

Перед началом работы приложения и в Twitter, и во многих других сервисах, сначала следует зарегистрировать приложение, в частности, такая регистрация может быть автоматической, она может подразумевать модерацию, требовать ввода каких-либо сведений о приложении. Регистрация приложения в сервисах преследует цель идентификации приложения. В данном случае процедура регистрации выполняется в разделе Manage & create your application (Управление приложением и создание приложений), для запуска процесса создания приложения служит кнопка Create a new application (Создать новое приложение). После успешного завершения регистрации и получения данных, которые приложение будет использовать для авторизации в сервисе (в частности, это - Consumer key и Consumer secret), можно приступать к изучению API.

Не все вызовы веб-сервисов требуют авторизации пользователя. Например, к некоторым механизмам Twitter можно обращаться напрямую, без авторизации.

Для того, чтобы приступить к созданию приложения для Facebook, нужно начать со страницы https://developers.facebook.com. Зарегистрировавшись в этой службе и создав новое приложение, вы получите данные, необходимые для дальнейшей работы, в частности, это AppID/API key и App Secret. У сервиса Вконтакте так же есть страница документации для разработчиков, http://vk.com/developers.php#devstep1. Здесь используется та же процедура регистрации приложения. Интерфейс для программного доступа к службе имеется и у проекта Wikipedia, http://www.mediawiki.org/wiki/API:Main_page/ru.

Приступая к исследованию API веб-служб, следует учесть, что к разработке можно подходить двумя путями. Во-первых, можно самостоятельно изучить подробности реализации API, подготовить вспомогательные классы для работы с ним - для организации вызова API с нужными параметрами, организации разбора ответа сервера и обработки ошибок. Подобная работа может оказаться весьма трудоёмкой и в некоторых случаях, когда существуют доступные библиотеки сторонних разработчиков, предназначенные для работы с тем или иным сервисом, можно воспользоваться такими библиотеками.

Задания:

1. Изучить рекомендуемую литературу.
2. Подумайте, какие выгоды приложение, разработкой которого вы занимаетесь, может извлечь из работы с веб-сервисами, предоставляющими программный доступ к своим возможностям. Существует огромное количество таких сервисов, если известные вам сервисы не представляется возможным использовать в приложении, поработайте со списком веб-служб по этому адресу: http://www.programmableweb.com/apis. Здесь сервисы разделены по категориям, присутствуют краткие описания их API. Выберите как минимум 5 служб, которые могли бы представлять интерес для использования в вашем приложении и подготовьте обзорное сообщение по ним. Вы можете выбрать и большее количество служб - возможно, кому-нибудь из группы пригодится один из сервисов, рассмотренных вами.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Методика разработки приложений, которые могут взаимодействовать с вебслужбами, используя API этих служб.
2. Назначение общедоступных библиотек, разработанных для некоторых API служб.

Практическая работа № 3.24. Хранение данных на устройстве. Локальные базы данных

Цель работы: освоить технологию хранения данных на устройстве, освоить технологию работы с локальными базами данных на Windows Phone.

Так как операции, подразумевающие работу с устройствами постоянного хранения данных, сравнительно медленны, для такой работы используют асинхронные механизмы. Кроме того, если в нескольких местах приложения нужно пользоваться данными, которые могут быть считаны, например, из изолированного хранилища настроек, имеет смысл, для ускорения работы приложения, сократить число обращений к хранилищу до минимума. Это можно сделать, воспользовавшись общедоступными переменными, в которые считываются эти данные.

Приложения для Windows Phone 8 могут хранить реляционные данные в локальных базах данных, с помощью механизмов Linq to SQL обеспечивается объектное представление баз данных в приложениях, что позволяет организовать удобную работу с ними.

Задания:

1. Изучить рекомендуемую литературу.
2. Использование файлов, хранящихся в локальных папках приложения, позволяет сохранять сериализованные представления объектов и, соответственно, загружать их. Это позволяет сохранять состояние приложения, например, при выходе из него, и загружать его при запуске приложения, то есть, если это оправдано сценарием приложения, создать у пользователя впечатление о том, что приложение "работает" непрерывно, позволяя ему, не беспокоясь о потере данных, в любой момент работы выходить из приложения, выполнять другие задачи на телефоне, а потом снова запускать его и продолжать работу. Проанализируйте приложение, которое вы создаёте, с учетом возможности сохранения его состояния в перерывах между работой с ним пользователя.
3. Если подобная модель взаимодействия с пользователем подходит вашему приложению, подумайте, как сократить до минимума объем данных, которые нужно хранить между запусками приложения, а так же, если объем данных достаточно велик, о так называемом инкрементном сохранении данных приложения во время работы, а не только при вызове обработчиков жизненного цикла приложения. То же самое касается загрузки приложения - чем меньший объем данных ему приходится считывать и обрабатывать при запуске - тем лучше. Подготовьте отчёт о проделанной работе с описанием особенностей сохранения и восстановления данных приложения между сеансами работы.
4. Если приложение, созданием которого вы занимаетесь, подразумевает работу с данными, имеющими сложную структуру, рассмотрите возможность использования локальной базы данных для работы с ними. Кроме того, если приложение рассчитано на работу с локальной базой данных, это упростит, при необходимости, возможность перехода на использование облачной базы данных, работающей на платформе Windows Azure с использованием локальной базы данных в качестве локального кэша. Подготовьте отчёт по данному этапу работы, приведите в нём анализ данных, которые использует приложение и сделайте выводы о том, поможет ли использование базы данных улучшить структуру вашего приложения.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Основные приемы работы с локальными папками приложений, с файлами, и с изолированным хранилищем настроек.
2. Назначение асинхронных механизмов.
3. Назначение общедоступных переменных.
4. Методика разработки приложений, использующих базы данных.
5. Назначение реляционных данных в локальных базах данных.
6. Назначение механизмов Linq to SQL.

Практическая работа № 3.25. Разработка для Windows Azure

Цель работы: освоить технологию работы с Windows Azure.

Основа Windows Asure - дата-центры, размещенные по всему миру. Платформа Azure появилась в 2008-м году, она постоянно развивается, увеличивается количество и мощность дата-центров, появляются новые услуги, существующие услуги улучшаются. Фактически, используя платформу Azure разработчик может создавать широкий спектр приложений, может использовать большой набор программного обеспечения (такого, как операционные системы, средства разработки, системы управления базами данных). На самом деле, список возможностей Azure весьма широк, рекомендуется самостоятельно ознакомиться с их составом и особенностями на http://www.windowsazure.com. Служба официально доступна в России, существует немало русскоязычных описаний её функциональных возможностей. Интерфейс управления Azure так же русифицирован.

Для регистрации в службе, https://account.windowsazure.com, вам понадобится учетная запись Microsoft, мобильный телефон (для того, чтобы принять код подтверждения), и кредитная (или дебетовая) карта, необходимая для подтверждения личности. При регистрации можно выбрать вариант пробной бесплатной подписки. Бесплатное пробное использование рассчитано на 90 дней, каждый месяц бесплатного пробного периода предоставляется некоторый объем услуг. В частности, во время пробного бесплатна работа с мобильными службами (mobile services), которые представляют особый интерес для мобильных разработчиков, так как позволяют быстро создавать облачные серверные части для мобильных приложений.

Задания:

1. Изучить рекомендуемую литературу.
2. Подумайте над тем, как вы можете использовать мобильные службы Azure в приложении, разработкой которого вы занимаетесь, рассмотрите возможности использования облачной базы данных, работы с Push-уведомлениями, аутентификации пользователей с использованием различных провайдеров аутентификации в применении к вашему приложению и подготовьте отчёт о проделанной работе.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Технологии реализации работы с мобильными службами Windows Azure в приложениях для Windows Phone 8.
2. Назначение взаимодействия со службой, используя класс MobileServiceClient.

Практическая работа № 3.26. Сервисы Live Connect: SkyDrive

Цель работы: освоить технологию работы с сервисом хранения данных SkyDrive и получить сведения, необходимые для дальнейшей работы со службами Live Connect.

Набор служб Live Connect предоставляет разработчику следующие инструменты:

- API для взаимодействия с хранилищем данных SkyDrive, в котором можно работать с папками, и файлами, такими, как фотоснимки, видеоролики, и документы.

Файлы и папки можно создавать, удалять, перемещать, отправлять в облачное хранилище с устройства и загружать на устройство из облачного хранилища. API SkyDrive, кроме того, позволяет работать с комментарии и тегами.

- Доступ к Hotmail, который позволяет работать с контактами и календарями.
- Работу с Windows Live Messenger, который предназначен для обмена мгновенными сообщениями.

Для работы с этими службами, которая подразумевает предварительную проверку подлинности пользователя, используется сервис Live. Кроме того, этот сервис предоставляет сведения о профиле пользователя. Подробности о службах Live Connect можно узнать из документации: http://msdn.microsoft.com/ru-ru/library/live/ff621314.aspx.

Для разработки с использованием Live API следует загрузить и установить соответствующий SDK, http://msdn.microsoft.com/ru-ru/live/ff621310.aspx. Он предоставляет набор элементов управления и API, которые позволяют интегрировать приложения со службой единого входа (SSO, Single Sign On) учетных записей Microsoft (в общем случае это позволяет производить автоматическую аутентификацию пользователя в приложениях, которые поддерживают одного и того же провайдера аутентификации), работать со SkyDrive, Hotmail и Windows Live Messenger из приложений для Windows Phone и Windows 8.

Задания:

1. Изучить рекомендуемую литературу.
2. Проанализируйте сценарии работы с приложением, созданием которого вы занимаетесь. Выделите те из них, в реализации которых можно использовать SkyDrive. Подготовьте отчёт о проделанной работе.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Назначение сервисов Live Connect.
2. Назначение службы облачного хранения данных пользователей SkyDrive.
3. Технология работы с файлами, хранящимися в SkyDrive.

Практическая работа № 3.27. Многопоточное программирование

Цель работы: освоить технологию работы с классом BackgroundWorker.

Создавая простое приложение для Windows Phone, не выполняющее сложных вычислений, не обращающееся к веб-сервисам, не выполняющее работу с файлами, о выполнении каких-либо задач вне потока пользовательского интерфейса обычно не задумываются. Если мы, например, в обработчике события нажатия на кнопку захотим выполнить какое-то простое вычисление, или проверку какого-либо условия, вычислительная нагрузка от таких операций ляжет на поток пользовательского интерфейса. До тех пор, пока эта нагрузка незначительна, подобное не сказывается на скорости отклика приложения. Но если нужно выполнить какую-либо ресурсоёмкую операцию, это можно сделать и без дополнительных усилий по использованию для неё другого потока, однако, выполнение её в потоке пользовательского интерфейса приведет к заметным

задержкам. Как вы увидите в примере, который мы разберем в этой лабораторной работе, интерфейс может быть блокирован. То есть, приложение не реагирует на взаимодействие пользователя с элементами управления, анимация может либо приостановиться, либо работать рывками.

В предыдущих лабораторных работах мы сталкивались с вызовом асинхронных методов для выполнения задач, которые могут занимать потенциально достаточно большое время. При объявлении метода, в котором мы вызывали асинхронные операции, используется модификатор async, операция

вызывается с использованием оператора await. Это приводит к тому, что выполнение метода приостанавливается до тех пор, пока не будет получен ответ от вызываемой процедуры. Например - данные, загруженные с вебсервера. При этом использование async и await упрощает написание асинхронного кода, так как до появления этого механизма (это произошло в Visual Studio 2012) при вызове асинхронного метода нужно было предусматривать использование функции, которая вызывается при завершении работы метода и обрабатывает полученные результаты. Такой подход применим и сейчас, но он усложняет структуру кода, в то время как асинхронный код, написанный с использованием async и await, хотя и решает задачи асинхронной обработки, выглядит как синхронный.

До сих пор мы вызывали асинхронные методы каких-либо объектов, но нередко возникают задачи, для успешного решения которых нужно асинхронно выполнить произвольный код. То есть, выполнить этот код не в потоке пользовательского интерфейса, а в другом потоке. При таком подходе интерфейс продолжает реагировать на воздействия пользователя. Решить подобную задачу можно, например, с использованием класса B ackgroundW orker (http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker%28v=vs.95%29.aspx).

Задания:

1. Изучить рекомендуемую литературу.
2. Проанализируйте код приложения, которое вы разрабатываете, испытайте различные варианты работы с ним и рассмотрите возможность асинхронного исполнения участков кода, которые выполняют в потоке пользовательского интерфейса какие- либо ресурсоёмкие операции (например, обрабатывают большие массивы данных). Подготовьте отчёт, содержащий данные анализа кода приложения на предмет асинхронного выполнения, и, если это применимо к вашему приложению, примеры модифицированного кода.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Особенности работы с асинхронным кодом.
2. Назначение использования асинхронного кода.

Практическая работа № 3.28. Сенсорный пользовательский интерфейс

Цель работы: освоить методы работы с сенсорным экраном.

Устройства, работающие под управлением Windows Phone не оснащают аппаратной клавиатурой. Такая тенденция наблюдается сейчас повсеместно, поэтому особую важность во взаимодействии с пользователем приобретает экранная клавиатура устройства. Такая клавиатура вызывается автоматически, когда пользователь входит в режим редактирования полей, предусматривающих текстовый ввод. Кроме того, сейчас разработчик для платформы Windows Phone не может создавать собственные варианты клавиатуры, однако, это ограничение компенсируется большим набором вариантов встроенной в систему экранной клавиатуры.

Для того, чтобы задать тип клавиатуры, которая отображается при попытке пользователя редактировать содержимое элемента управления, поддерживающего ввод текста, нужно настроить свойство этого элемента InputScope (Область вводимых данных). Сделать это можно как в XAML-разметке, так и в программном коде (http://msdn.microsoft.com/en-us/library/windowsphone/develop/gg521152%28v=vs.105%29.aspx). Существует много вариантов такой клавиатуры. При заполнении этого свойства, например, для текстового поля, можно воспользоваться либо окном свойств, где представлен список доступных вариантов, либо, при заполнении его в XAML- коде - подсказкой IntelliSense.

Оборудование и материалы: для выполнения данной лабораторной работы необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: MS Word, Adobe Reader, и Visual Studio 2012.

Указания по технике безопасности: к выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания:

1. Изучить рекомендуемую литературу.
2. Проанализируйте пользовательский интерфейс приложения, разработкой которого вы занимаетесь, обращая внимание на то, какие именно данные пользователь вводит в текстовые поля, присутствующие в интерфейсе. Подберите такие значения свойства InputScope полей ввода, которые позволят пользователю наиболее удобно и быстро вводить данные. Подумайте, как использование жесты позволит улучшить опыт взаимодействия пользователя и приложения. Подготовьте отчёт.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Способы настройки экранной клавиатуры с помощью установки свойства InputScope текстового блока.
2. Назначение использования экранной клавиатуры.
3. Методика обработки жестов.

Практическая работа № 3.29. Работа с датчиками, определение местоположения

Цель работы: освоить методы работы со встроенными датчиками.

При разработке для Windows Phone можно использовать показания встроенных датчиков устройства, сведения о них можно найти в обзорном материале по пространству имен Windows.Devices.Sensors (http://msdn.microsoft.com/en-us/library/windowsphone/ develop/br206408.aspx). Один из доступных датчиков - акселерометр, он позволяет организовывать управление программами путем перемещения телефона в пространстве.

При работе с акселерометром данные, которые можно от него получить, представляют сведения об ускорении устройства в гравитационных единицах. Например, (1,0,0). Если сопоставить эти данные с телефоном, то, если расположить телефон кнопками, расположенными под экраном, вниз, лицевой стороной к наблюдателю, окажется, что ось Y проходит вдоль длинной стороны экрана (положительное направление оси - вверх), X - вдоль короткой (положительное направление оси - вправо), ось Z располагается перпендикулярно экрану (положительное направление - в сторону наблюдателя). Это остаётся справедливым и при поворотах телефона.

Когда телефон неподвижен, акселерометр, регистрирует силу земного притяжения, что отражается в его показаниях (они, по соответствующей оси, близки к 1), перемещения телефона в пространстве приводят к изменению показаний.

Таким образом, оказывается, например, если телефон лежит на горизонтальной поверхности неподвижно, экраном вверх, показания акселерометра выглядят как (0, 0, -1). Если расположить телефон вертикально, кнопками вниз, мы имеем в показаниях (0, -1, 0), перевернув телефон "вверх ногами" - (0, 1, 0). Если повернуть телефон на 90° против часовой стрелки - мы получим (-1,0,0). Акселерометр очень чувствителен, даже при неподвижном устройстве показания не бывают в точности равными идеальным, к тому же, они колеблются в небольших пределах.

Задания:

1. Изучить рекомендуемую литературу.
2. Подумайте, как использование встроенных датчиков устройства может расширить возможности пользователя по управлению приложением, которое вы разрабатываете. Кроме того, проанализируйте возможности использования в приложении сервиса определения местоположения.

Подготовьте отчет о проделанной работе.

3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Методика работы с акселерометром и системной определения местоположения в Windows Phone.
2. Назначение акселерометра.
3. Назначение системы определения местоположения.

Практическая работа № 3.30. Распознавание и синтез речи, работа с камерой

Цель работы: освоить технологию хранения данных на устройстве.

Возможности платформы Windows Phone 8 по распознаванию и синтезу речи можно использовать для организации взаимодействия приложения с пользователем. Так, пользователь может отдавать приложению голосовые команды, которые приложение может воспринять благодаря подсистеме распознавания речи. Приложение, в свою очередь, используя функцию синтеза речи, преобразования текста в речь - информировать пользователя о каких-либо событиях.

Задания:

1. Изучить рекомендуемую литературу.
2. Рассмотрите возможность интеграции в приложение, которое вы разрабатываете, функций распознавания и синтеза речи, проанализируйте сценарии работы с приложением, в которых эти возможности наиболее полезны. Если вы разрабатываете приложение для работы с фотографиями, рассмотрите его реализацию с использованием функциональности фотоприложения. Подготовьте отчёт.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman; интервал - полуторный, Абзацный отступ - 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 - Создание подсистемы) и ссылку на него в тексте

Контрольные вопросы:

1. Методика работы с системой распознавания и синтеза речи в Windows Phone.
2. Возможности приложений по взаимодействию с пользователем.
3. Концепция фотоприложений.

МДК.01.04 Системное программирование

Практическая работа №1 Использование потоков

Цель работы: *ознакомление с общими принципами построения программы на языке ассемблер.*

3. Ход работы:

1. Изучить теоретическую часть (дополнительная информация в файле Структура COM и EXE файлов.docx)
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

&Теоретическая часть:

Структура ассемблерной программы

Каждый язык программирования имеет свои особенности. Язык ассемблера - не исключение. Традиционно первая программа выводит приветственное сообщение на экран ‘Hello World’.

В отличие от многих современных языков программирования в ассемблерной программе каждая команда располагается на ОТДЕЛЬНОЙ СТРОКЕ. Нельзя разместить несколько команд на одной строке. Не принято, также, разбивать одну команду на несколько строк. Язык ассемблера является РЕГИСТРОНЕЧУВСТВИТЕЛЬНЫМ.Т. е. в большинстве случаев нет разницы между большими и малыми буквами. Команда может быть ДИРЕКТИВОЙ - указанием транслятору. Они выполняются в процессе превращения программы в машинный код. Многие директивы начинаются с точки. Для удобства чтения программы они обычно пишутся БОЛЬШИМИ БУКВАМИ. Кроме директив еще бывают ИНСТРУКЦИИ - команды процессору. Именно они и будут составлять машинный код программы.

Процесс обработки программы на языке ассемблера

Весь процесс технического создания ассемблерной программы можно разбить на 4 шага (исключены этапы создания алгоритма, выбора структур данных и т.д.).

Набор программы в текстовом редакторе и сохранение ее в отдельном файле. Каждый файл имеет имя и тип, называемый иногда расширением. Тип в основном используется для определения назначения файла. Например, программа на C имеет тип C, на Pascal - PAS, на языке ассемблера - ASM.

Обработка текста программы транслятором. На этом этапе текст превращается в машинный код, называемый объектным. Кроме того, есть возможность получить листинг программы, содержащий кроме текста программы различную дополнительную информацию и таблицы, созданные транслятором. Тип объектного файла - OBJ, файла листинга - LST. Этот этап называется ТРАНСЛЯЦИЕЙ.

Обработка полученного объектного кода компоновщиком. Тут программа "привязывается" к конкретным условиям выполнения на ЭВМ. Полученный машинный код называется выполняемым. Кроме того, обычно получается карта загрузки программы

в ОЗУ. Выполняемый файл имеет тип EXE, карта загрузки - MAP. Этот этап называется КОМПОНОВКОЙ или ЛИНКОВКОЙ.
Запуск программы. Если программа работает не совсем корректно, перед этим может присутствовать этап ОТЛАДКИ программы при помощи специальной программы - отладчика. При нахождении ошибки приходится проводить коррекцию программы, возвращаясь к шагу 1. Таким образом, процесс создания ассемблерной программы можно изобразить в виде следующей схемы. Конечной целью, напомним, является работоспособный выполняемый файл HELLO. EXE.

Особенности создания ассемблерной программы в среде эмулятора EMU8086

Этот программный продукт содержит все необходимое для создания программы на языке Assembler.
Пакет Emu8086 сочетает в себе продвинутый текстовый редактор, assembler, disassembler, эмулятор программного обеспечения (Виртуальную машину) с пошаговым отладчиком, примеры.

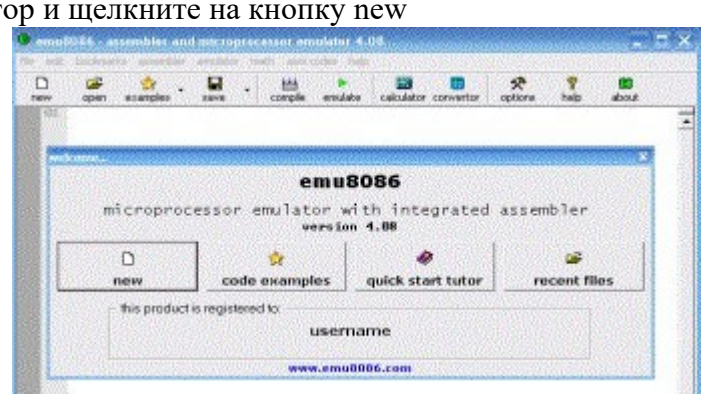
Правила оформления ассемблерных программ

При наборе программ на языке ассемблера придерживайтесь следующих правил:
директивы набирайте большими буквами, инструкции - малыми;
пишите текст широко - не скупердяйничайте;
не выходите за край экрана, т.е. не делайте текст шире 80 знаков - его не удобно будет редактировать и печатать;
для отступов пользуйтесь табуляцией (клавиша TAB);
блоки комментариев задавайте с одинаковым отступом. Оптимальной считается такая строка:
movax,<пробел>bx< (1-3) TAB>; <пробел>текст комментария
Количество табуляций перед комментарием определяется длиной аргументов команды и может быть от 1 до 3. По мере знакомства с синтаксисом языка будут приводиться дополнительные правила.

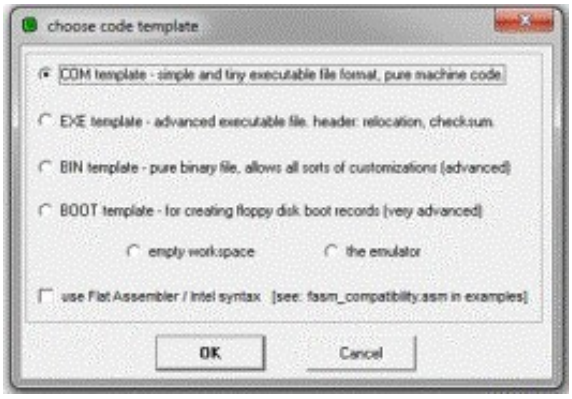
Г**Задание 1.** Работа с эмулятором Emu8086

Emu8086 сочетает в себе мощный редактор исходного кода, ассемблер, дизассемблер, программный эмулятор (виртуальный ПК) с отладчиком и поэтапное обучение. Эта программа компилирует исходный код и выполняет его с помощью эмулятора шаг за шагом.

1. Запустите эмулятор и щелкните на кнопку new

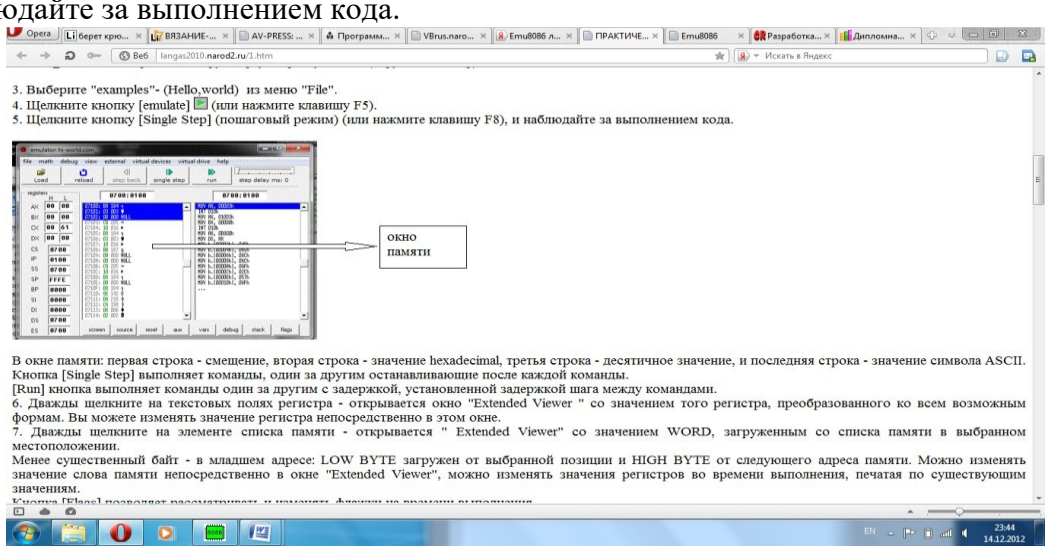


2. Выберите тип исполняемого файла:



Директивы, определяющие тип исполнимого файла:
#MAKE_COM#
#MAKE_BIN#
#MAKE_BOOT#
#MAKE_EXE#
Вы можете вставить эти директивы в исходный код для определения нужного вам типа исполнимого файла.
Описание типов исполнимых файлов:
#MAKE_COM# - самый старый и самый простой формат исполнимого файла. Такие файлы загружаются с префиксом 100h (256 байтов).
#MAKE_EXE# - более "продвинутый" формат исполнимого файла. Не ограничены размер и количество сегментов.
#MAKE_BIN# - простой исполнимый файл.
#MAKE_BOOT# - эта директива копирует первую дорожку дискеты (загрузочный сектор).

3. Выберите "examples"- (Hello,world) из меню "File".
4. Щелкните кнопку [emulate] (или нажмите клавишу F5).
5. Щелкните кнопку [Single Step] (пошаговый режим) (или нажмите клавишу F8), и наблюдайте за выполнением кода.



В окне памяти: первая строка - смещение, вторая строка - значение hexadecimal, третья строка - десятичное значение, и последняя строка - значение символа ASCII.
Кнопка [Single Step] выполняет команды, один за другим останавливающие после каждой команды.
[Run] кнопка выполняет команды один за другим с задержкой, установленной задержкой шага между командами.

6. Дважды щелкните на текстовых полях регистра - открывается окно "Extended Viewer " со значением того регистра, преобразованного ко всем возможным формам. Вы можете изменять значение регистра непосредственно в этом окне.

7. Дважды щелкните на элементе списка памяти - открывается " Extended Viewer" со значением WORD, загруженным со списка памяти в выбранном местоположении. Менее существенный байт - в младшем адресе: LOW BYTE загружен от выбранной позиции и HIGH BYTE от следующего адреса памяти. Можно изменять значение слова памяти непосредственно в окне "Extended Viewer", можно изменять значения регистров во времени выполнения, печатая по существующим значениям.

Кнопка [Flags] позволяет рассматривать и изменять флажки на времени выполнения.

В окне эмулятора вы можете запустить вашу программу на выполнение целиком (кнопка RUN) либо в пошаговом режиме (кнопка SINGLE STEP). Пошаговый режим удобен для отладки. Ну а мы сейчас запустим программу на выполнение кнопкой RUN. После этого (если вы не сделали ошибок в тексте программы) вы увидите сообщение о завершении программы (рис. 1.3). Здесь вам сообщают о том, что программа передала управление операционной системе, то есть программа была успешно завершена. Нажмите кнопку OK в этом окне и вы увидите, наконец, результат работы вашей первой программы на языке ассемблера (рис. 1.4).

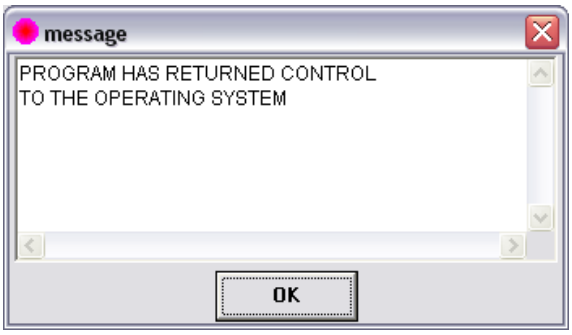


Рис. 1.3. Сообщение о завершении программы.

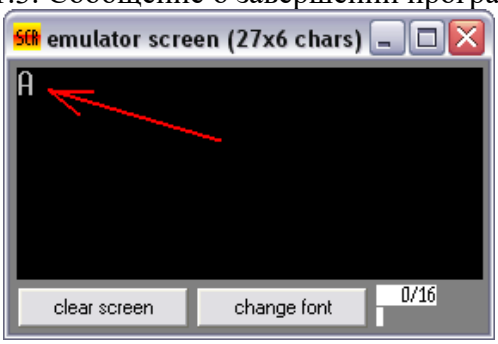
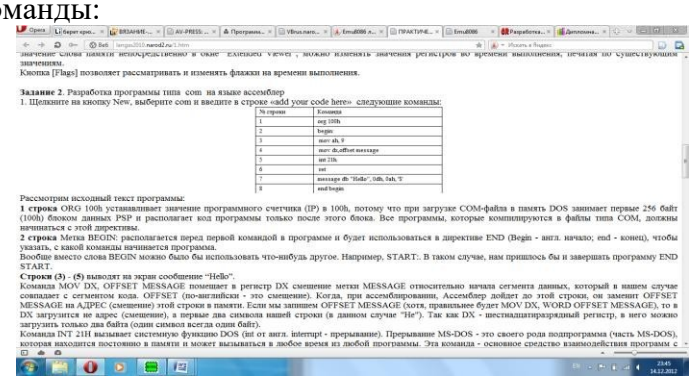


Рис. 1.4. Ваша первая программа выполнена.

Г**Задание 2.** Разработка программы типа com на языке ассемблер

1. Щелкните на кнопку New, выберите com и введите в строке «add your code here» следующие команды:



Рассмотрим исходный текст программы:

1 строка ORG 100h устанавливает значение программного счетчика (IP) в 100h, потому что при загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока. Все программы, которые компилируются в файлы типа COM, должны начинаться с этой директивы.

2 строка Метка BEGIN: располагается перед первой командой в программе и будет использоваться в директиве END (Begin - англ. начало; end - конец), чтобы указать, с какой команды начинается программа.

Вообще вместо слова BEGIN можно было бы использовать что-нибудь другое. Например, START:. В таком случае, нам пришлось бы и завершать программу END START.

Строки (3) - (5) выводят на экран сообщение “Hello”.

Команда MOV DX, OFFSET MESSAGE помещает в регистр DX смещение метки MESSAGE относительно начала сегмента данных, который в нашем случае совпадает с сегментом кода. OFFSET (по-английски - это смещение). Когда, при ассемблировании, Ассемблер дойдет до этой строки, он заменит OFFSET MESSAGE на АДРЕС (смещение) этой строки в памяти. Если мы запишем OFFSET MESSAGE (хотя, правильнее будет MOV DX, WORD OFFSET MESSAGE), то в DX загрузится не адрес (смещение), а первые два символа нашей строки (в данном случае "Не"). Так как DX - шестнадцатиразрядный регистр, в него можно загрузить только два байта (один символ всегда один байт).

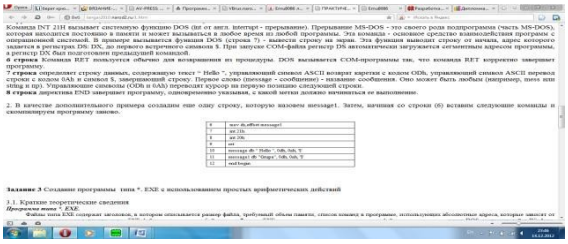
Команда INT 21H вызывает системную функцию DOS (int от англ. interrupt - прерывание). Прерывание MS-DOS - это своего рода подпрограмма (часть MS-DOS), которая находится постоянно в памяти и может вызываться в любое время из любой программы. Эта команда - основное средство взаимодействия программ с операционной системой. В примере вызывается функция DOS (строка 7) - вывести строку на экран. Эта функция выводит строку от начала, адрес которого задается в регистрах DS: DX, до первого встречного символа $. При запуске COM-файла регистр DS автоматически загружается сегментным адресом программы, а регистр DX был подготовлен предыдущей командой.

6 строка Команда RET пользуется обычно для возвращения из процедуры. DOS вызывается COM-программы так, что команда RET корректно завершает программу.

7 строка определяет строку данных, содержащую текст “ Hello ", управляющий символ ASCII возврат каретки с кодом ODh, управляющий символ ASCII перевод строки с кодом 0Ah и символ $, завершающий строку. Первое слово (message - сообщение) - название сообщения. Оно может быть любым (например, mess или string и пр). Управляющие символы (ODh и 0Ah) переводят курсор на первую позицию следующей строки.

8 строка директива END завершает программу, одновременно указывая, с какой метки должно начинаться ее выполнение.

2. В качестве дополнительного примера создадим еще одну строку, которую назовем message1. Затем, начиная со строки (6) вставим следующие команды и скомпилируем программу заново.



Г**Задание 3**. Создание программы типа *. EXE с использованием простых арифметических действий

Простые арифметические операторы.

4. **1. Сложение.**

Команда ADD (Addition - сложение (гл. to add - сложить)) осуществляет сложение первого и второго операндов. Исходное значение первого операнда (приемника) теряется, замещаясь результатом сложения. Второй операнд не изменяется. В качестве первого операнда команды ADD можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака. Команду ADD можно использовать для

сложения как обычных целых чисел, так и двоично-десятичных (с использованием регистра AX для хранения результата). Команда воздействует на флаги OF, SF, ZF, AF, PF и CF.

Примеры:

mov al,10; загружаем в регистр AL число 10
add al,15; al = 25; al - приемник, 15 - источник
mov ax,25000; загружаем в регистр AX число 25000
add ax,10000; ax = 35000; ax - приемник, 10000 - источник
mov cx, 200; загружаем в регистр CX число 200
mov bx,760;a в регистр BX - 760
add cx,bx; cx = 960, bx = 760 (bx не меняется); cx - приемник, bx - источник

2 Вычитание.

Команда SUB (Subtraction - вычитание) вычитает второй операнд (источник) из первого (приемника) и помещает результат на место первого операнда. Исходное значение первого операнда (уменьшаемое) теряется. Таким образом, если команду вычитания записать в общем виде

SUB операнд1, операнд2
то ее действие можно условно изобразить следующим образом:
операнд1 - операнд2 - > операнд1

В качестве первого операнда можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака. Команда воздействует на флаги OF, SF, ZF, AF, PF и CF.

Примеры:

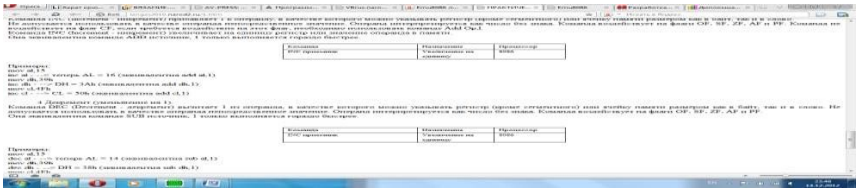
mov al,10
sub al,7 - --> al = 3; al - приемник, 7 - источник
mov ax,25000
sub ax,10000 - --> ax = 15000; ax - приемник, 10000 - источник
mov cx,100
mov bx,15
sub cx,bx - --> cx = 85, bx = 15 (bx не меняется); cx - приемник, bx - источник

3 Инкремент (увеличение на 1).

Команда INC (Increment - инкремент) прибавляет 1 к операнду, в качестве которого можно указывать регистр (кроме сегментного) или ячейку памяти размером как в байт, так и в слово. Не допускается использовать в качестве операнда непосредственное значение. Операнд интерпретируется как число без знака. Команда воздействует на флаги OF, SF, ZF, AF и PF. Команда не воздействует на флаг CF; если требуется воздействие на этот флаг, необходимо использовать команду Add Op,l.

Команда INC (Increment - инкремент) увеличивает на единицу регистр или значение операнда в памяти.

Она эквивалентна команде ADD источник, 1 только выполняется гораздо быстрее.

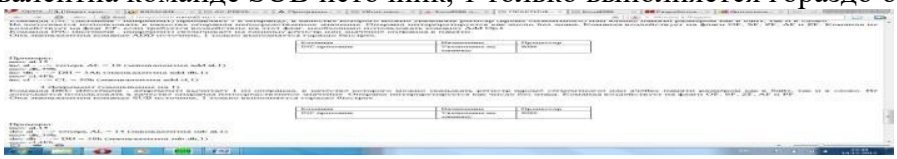


Примеры:

mov al,15
inc al - --> теперь AL = 16 (эквивалентна add al,1)
mov dh,39h
inc dh - --> DH = 3Ah (эквивалентна add dh,1)
mov cl,4Fh
inc cl - --> CL = 50h (эквивалентна add cl,1)

4 Декремент (уменьшение на 1).

Команда DEC (Decrement - декремент) вычитает 1 из операнда, в качестве которого можно указывать регистр (кроме сегментного) или ячейку памяти размером как в байт, так и в слово. Не допускается использовать в качестве операнда непосредственное значение.
Операнд интерпретируется как число без знака. Команда воздействует на флаги OF, SF, ZF, AF и PF.
Она эквивалентна команде SUB источник, 1 только выполняется гораздо быстрее.



Примеры:

mov al,15
dec al - --> теперь AL = 14 (эквивалентна sub al,1)
mov dh,39h
dec dh - --> DH = 38h (эквивалентна sub dh,1)
mov cl,4Fh
dec cl - --> CL = 4Dh (эквивалентна sub cl,1)

3. 2. Создание программ производящих сложение и вычитание.

1. Щелкните на кнопку New, выберите Exe и введите в строке «add your code here» следующие команды для расчета выражения:
((5 + (AL – AH)) – BH) + BL где AL=9, BL=3, AH=4, BH=6.

: (№ + AL) – (BH + AH) – BL

```
19 mov al, 9
20 mov ah, 4
21 sub al, ah
22 mov dl, 5
23 add al, dl
24 mov bh, 6
25 sub al, bh
26 mov bl, 3
27 add al, bl
```

Для вывода результирующего значения в 10-ричной системе счисления добавьте следующий код:

```
mov bx, 0
mov bl, al
 mov ax, bx
push -1
mov cx,10
l:mov dx,0
div cx
push dx
cmp ax,0
jne l
mov ah,2h
l2:pop dx
cmp dx,-1
je ex
add dl,'0'
int 21h
jmp l2
ex:mov ax,4c00h
int 21h
end start
```

5. ГКонтрольные задания:

Произвести расчет выражений:

1) (№ + AL) – (BH + AH) – BL
2) ((BH + (№ – AH)) – BL) + AL

где № - порядковый номер по журналу, AL=9, BL=3, AH=4, BH=6.

6. Контрольные вопросы:

1. Характеристика структуры файла типа *.com?
2. Какова структура ассемблерной программы?
3. С какой целью в код программы на ассемблере для DOS вводится строка ORG 100h?
4. Назначение команды MOV?
5. Прерывания 21h и 20h. Назначение?
6. Сущность и целесообразность использования команды RET вместо прерывания 20h?
6. Символ ‘$’ методика применения?
7. Связка “BEGIN: - END BEGIN". Правила применения?
8. Каковы этапы получения выполняемого файла?

Практическая работа №2 Обмен данными

иже приведен пример текста программы, которая на языке ассемблер в заданном массиве определяет элемент с максимальной величиной.

```
      SGSTACK SEGMENT PARA STACK 'STACK'
       DB 32 DUP(?)
SGSTACK ENDS

      DATA SEGMENT PARA PUBLIC 'DATA'
       MAX DW ?
 ARRAY  DW  10H,  20H,  30H,  0D0H,0A0H
DATA ENDS

      CODE   SEGMENT   PARA   PUBLIC   'CODE'
        ASSUME CS:CODE, DS:DATA, SS:SGSTACK

START: MOV AX, DATA ; загрузить в DS
        MOV DS, AX              ;селектор сегмента данных
        LEA BX, ARRAY           ; загрузить в BX начальный адрес массива
        MOV CX, 4               ; инициализировать счетчик
        MOV AX, [BX]            ; инициализировать начальное значение max

      CYCLE: ADD BX, 2        ; перейти к следующему элементу массива
        CMP [BX], AX          ; сравнить два значения
        JBE BE                ; переход если равен или ниже
        MOV AX, [BX]          ; сохранить большее значение
      BE: LOOP CYCLE          ; проверка на выход из цикла (--CX при CX=0)
        MOV MAX, AX           ; сохранение максимального значения

      EXIT: XOR AL, AL        ; выход в OS
        MOV AH, 4CH
  INT  21H  CODE
ENDS          END
START
```

Порядок выполнения работ

Для выполнения лабораторной работы необходимо:

1. Получить вариант задание у преподавателя из таблицы
2. Составить программу согласно заданному варианту
3. Получить файл с исходным текстом программы в EXE и COM формате
4. Оттранслировать, отладить программу. Изучить листинг программы.
5. Скомпоновать выполняемый файл, изучить карту загрузки (порядок следования сегментов, их размеры и относительные адреса)
6. Запустить программу под отладчиком.
7. Оформить отчет

Практическая работа №3 Сетевое программирование сокетов

7. Цель работы

Изучить:

- возможности реализации архитектуры клиент-сервер на основе интерфейса сокетов Windows Sockets API;
- типы сокетов TCP/IP;
- основные методики и API-функции для разработки сетевых приложений с использованием Winsock.

8. Постановка задачи

1. Изучить методические указания к лабораторной работе, материалы лекций и рекомендуемую литературу.
2. Разработать консольное клиент-серверное приложение, демонстрирующее взаимодействие на основе потоковых сокетов.
3. Разработать консольное клиент-серверное приложение, демонстрирующее взаимодействие на основе дейтаграммных сокетов.

9. Методические указания

1. Понятие сокета

Сокет (Socket - гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью.

Соединяя вместе два сокета, можно передавать данные между разными процессами (локальными и удаленными). Реализация сокетов обеспечивает инкапсуляцию протоколов сетевого и транспортного уровней.

Интерфейс, используемый приложением при взаимодействии с программным обеспечением транспортного протокола, называется ***интерфейсом прикладного программирования*** (Application Programming Interface - API). API интерфейс определяет набор операций, которые могут быть выполнены приложением при взаимодействии с программным обеспечением протокола.

Функции прикладного программного интерфейса сокетов (Sockets API) обеспечивают идентификацию конечных точек соединения, установку соединения, отправку сообщений, ожидание входящих сообщений, разрыв соединения и обработку ошибок.

2. Протоколы и семейства адресов

Важнейшим преимуществом сокетов Windows является предоставление единого независимого интерфейса сетевого программирования (Sockets API) для различных сетевых протоколов.

Платформы Win32 поддерживают разнообразные сетевые стеки протоколов : TCP/IP, IPX/SPX, NetBIOS/SMB, AppleTalk, ATM, Infrared Sockets. Каждому из них соответствует свое семейство адресов сокетов. Напрмер, TCP/IP соответствует семейство адресов AF_INET, IPX/SPX – AF_NS, ATM –AF_ATM и т.д.

Семейство адресов – важнейший параметр сокета. Он указывает используемый в настоящее время протокол и ограничивает применение других параметров сокета.

Мы рассмотрим адресацию сокетов только для стека протоколов TCP/IP, как самого распространенного на сегодняшний день.

Адрес сокета при использовании протоколов TCP/IP задает следующий набор значений:

- номер сети;
- номер конечного узла;
- номер порта прикладной службы.

3. Инициализация Winsock

Перед вызовом любой функции Winsock необходимо загрузить соответствующую версию библиотеки Winsock. Для использования в приложении Winsock 2 необходимо подключить библиотеку Ws2_32.lib и заголовочный файл Winsock2.h.

Имена новых или обновленных API-функций, добавленные в Winsock 2, начинаются с префикса WSA.

Инициализацию Winsock выполняет функция *WSAStartup:*

int WSAStartup(

```
WORD        wVersionRequested,
LPWSADATA  IpWSAData);
```

Параметр *wVersionRequested* задает версию загружаемой библиотеки Winsock. На современных платформах Win32 используется версия 2.2. Для получения значения параметра *wVersionRequested* можно использовать макрос *MAKEWORD(2, 2)* либо значение 0x0202.

Параметр *IpWSAData* — указатель на структуру *LPWSADATA,* которая при вызове функции *WSAStartup* заполняется сведениями о версии загружаемой библиотеки.

По завершении работы с библиотекой Winsock необходимо вызвать функцию *WSACleanup* для выгрузки библиотеки и освобождения ресурсов:

```
int WSACleanup (void);
```

4. Адресация сокетов для протокола IP

Для задания IP-адреса и номера порта службы используется структура *SOCKADDR_IN.* Она определена в include-файле in.h следующим образом*:*

```
struct sockaddr_in {
        short           sin_family;
        u_short         sin_port;
        struct in_addr  sin_addr;
        char            sin_zero[8];
        };
```

Поле *sin_family* при использовании семейства адресов IP должно быть равно *AF_INET*.

Поле *sin_port* задает, какой коммуникационный порт будет использован для идентификации службы.

Поле *sin_addr* структуры *SOCKADDR_IN* хранит IP-адрес в 4-байтном виде с типом unsigned long int. В зависимости от того, как это поле использовано, оно может представлять и локальный, и удаленный IP-адрес.

Поле *sin_zero* играет роль заполнителя, чтобы структура *SOCKADDR_IN* по размеру равнялась структуре *SOCKADDR.*

5. Специальные IP-адреса

Специальный адрес *INADDR_ANY (0.0.0.0)* позволяет приложению слушать клиента через любой сетевой интерфейс на несущем компьютере. Обычно приложения сервера используют этот адрес, чтобы привязать сокет к локальному интерфейсу для прослушивания соединений. Если на компьютере несколько сетевых адаптеров, то этот адрес позволит отдельному приложению получать отклики от нескольких интерфейсов.

Второй специальный адрес – *INADDR_BROADCAST (255.255.255.255),* позволяет широковещательно рассылать пакеты по IP – сети. Для его использования необходимо в приложении задать параметр сокета *SO_BROADCAST.*

6. Порядок байт

В памяти компьютера IP-адрес и номер порта представляются в системном порядке (host-byte-order). Для процессоров Intel Pentium это порядок от менее значимого к более значимому байту (обратный). Стандарты Internet требуют, чтобы многобайтные значения передавались от старшего байта к младшему, что называется сетевым порядком (network- byte order) или прямым порядком следования байтов. Поэтому существует необходимость преобразования чисел из одной формы в другую. Для этого предназначен целый ряд функций. Например, функции htonl (Host TO Network Long), WSAHtonl, htons (Host TO Network Short), WSAHtons преобразуют числа из системного порядка в сетевой. Функции ntohl, WSANtohl, ntohs, WSANtohs решают обратную задачу (из сетевого в системный).

Полезная вспомогательная функция *inet_addr* преобразует IP-адрес из точечно-десятичной нотации в 32-битное длинное целое без знака с сетевым порядком следования байт:

unsigned long inet_addr(const char FAR *cp);

Поле *cp* - строка, заканчивающаяся нулевым символом, в которой задается IP-адрес в точечно-десятичной нотации.

7. Разрешение имен

В Winsock предусмотрено две функции для разрешения имени в IP-адрес.

Функции gethostbyname и WSAAsyncGetHostByName отыскивают в базе данных узла сведения об узле, соответствующие его имени. Обе функции возвращают структуру *HOSTENT*:

```
struct hostent
{
    char FAR *          h_name;
    char  FAR  *  FAR  *  h_aliases;
    short               h_addrtype;
    short               h_length;
    char FAR * FAR *    h_addr_list;
};
```

Поле *h_name* является официальным именем узла. Если в сети используется *доменная система имен* (Domain Name System, DNS), в качестве имени сервера будет возвращено полное имя домена (Fully Qualified Domain Name, FQDN). Если в сети для разрешения имен применяется локальный файл (hosts, lmhosts) - это первая запись после IP- адреса.

Поле *h_aliases* – массив дополнительных имен узла, завершающийся нулем.

Поле *h_addrtype* – возвращаемое семейство адресов.

Поле *h_length* определяет длину в байтах каждого адреса из поля *h_addr_list*.

Поле *h_addr_list* – массив, завершающийся 0 и содержащий IP-адреса узла. (Узел может иметь несколько IP-адресов). Каждый адрес в этом массиве представлен в сетевом порядке. Обычно приложение использует первый адрес из массива. Впрочем, при получении нескольких адресов, приложение должно выбирать адрес случайным образом из числа доступных, а не упорно использовать первый.

API–функция *gethostbyname* определена так:

```
struct hostent FAR  *gethostbyname  (const char  FAR  *name);
```

Параметр *name* представляет дружественное имя искомого узла. При успешном выполнении функции возвращается указатель на структуру HOSTENT, которая хранится в системной памяти. Приложение не должно полагать, что эти сведения непременно статичны. Поскольку эта память обслуживается системой, оно не должно освобождать возвращенную структуру.

WSAAsyncGetHostByName – асинхронная версия функции *gethostbyname*, оповещающая приложение о завершении своего выполнения с помощью сообщений Windows:

```
HANDLE WSAAsyncGetHostByName (
        HWND hWnd,
        unsigned  int  wMsg,
        const char FAR *name,
        char FAR * buf,
        int buflen);
```

Параметры *hWind* – дескриптор окна, которое получит сообщение по завершении выполнения асинхронного запроса. Параметр *wMsg* –Windows-сообщение, которое будет возвращено по завершении выполнения асинхронного запроса. Параметр *name* - дружественное имя искомого узла. Параметр *buf* – указатель на область данных, куда помещается *HOSTENT*. Этот буфер должен быть больше структуры *HOSTENT* и иметь размер, определенный в MAXGETHOSTSTRUCT.

Для преобразования IP-адреса в имя узла используются функции *gethostbyaddr* и *WSAAsyncGetHostByAddr*. Функция *gethostbyaddr* определена так:

```
struct  HOSTENT  FAR * gethostbyaddr(
        const char FAR * addr,
        int len,
        int type);
```

Параметр *addr* – указатель на IP–адрес в сетевом порядке.

Параметр *len* задает длину параметра *addr* в байтах.

Параметр *type* должен иметь значение AF_INET (IP-адрес).

Функция *WSAAsyncGetHostByAddr* – асинхронная версия функции *gethostbyaddr*.

8. Номера портов

Приложения должны быть внимательны при выборе номера порта, поскольку некоторые доступные порты зарезервированы для использования популярными службами, такими как FTP, HTTP и т.д. Эти порты обслуживаются и распределяются центром Internet Assigned Numbers Authority (IANA), их описание содержится в RFC 1700.

Номера портов разделяются на 3 категории (стандартные, зарегистрированные и динамические и/или частные):

- Номера от 0 до 1023 зарезервированы для стандартных служб.
- Порты с номерами от 1024 до 49151 являются регистрируемыми. Они используются для различных целей.
- Порты с номерами от 49152 до 65535 представляют собой динамические и частные порты.

Во избежание накладок с портами, уже занятыми системой или другим приложением, ваша программа должна выбирать порты, начиная с 1024. Можно вместо конкретного номера порта задать 0, тогда система сама выберет произвольный неиспользуемый в данный момент номер.

Узнать номера портов, используемых стандартными службами, можно вызвав функцию *getservbyname* или *WSAAsyncGetServByName*. Эти функции извлекают статическую информацию из файла services, расположенного в папке

%WINDOWS%\System32\Drivers\Etc

Функция *getservbyname* определена так:

```
struct servent FAR * getservbyname(
      const char FAR *name,
      const char FAR *proto);
```

Параметр *name* представляет имя искомой службы.

Параметр *proto* ссылается на строку, указывающую протокол, под которым зарегистрирована служба из параметра *name*.

Структура servent имеет системный порядок следования байт.

Функция *WSAAsyncGetServByName* – асинхронная версия *getservbyname.*

9. Типы сокетов

Существуют три основных типа сокетов: потоковые, дейтаграммные и сырые.

Потоковые сокеты – это сокеты с установлением соединения, состоящие из потока байтов, который может быть двунаправленным. Т.е. через такую конечную точку приложение может и передавать, и получать данные. Потоковый сокет гарантирует обнаружение и исправление ошибок, обрабатывает доставку и сохраняет последовательность данных. Он подходит для передачи больших объемов данных, поскольку в этом случае накладные расходы, связанные с установлением соединения, незначительны по сравнению со временем передачи самого сообщения. Качество передачи достигается за счет использования протокола TCP.

Дейтаграммные сокеты – это сокеты без установления соединения, не обеспечивающие надежность при передаче. Применяются для приложений, когда

неприемлемы затраты времени, связанные с установлением явного соединения. Для передачи данных используется протокол UDP.

Сырые сокеты (необрабатываемые, простые) – это сокеты, которые принимают пакеты сетевого уровня в обход протоколов транспортного уровня и отправляют их непосредственно приложению.

10. Коммуникационная модель клиент-сервер

Приложение, использующее сокеты, состоит из распределенной программы, исполняемой на обоих концах канала связи. Программу, инициирующую передачу, называют ***клиентом***. Приложение на другом конце соединения, называемое ***сервером***, представляет собой модуль, пассивно ожидающий входящих запросов на установку соединений от удаленных клиентов. Как правило, серверное приложение загружается при запуске системы и активно прослушивает свой порт, ожидая входящих соединений. Клиентские приложения пытаются установить соединение с сервером, после чего начинается обмен данными. По завершении сеанса связи клиент, как правило, разрывает соединение. На рисунке представлена базовая модель взаимодействия потоковых сокетов.

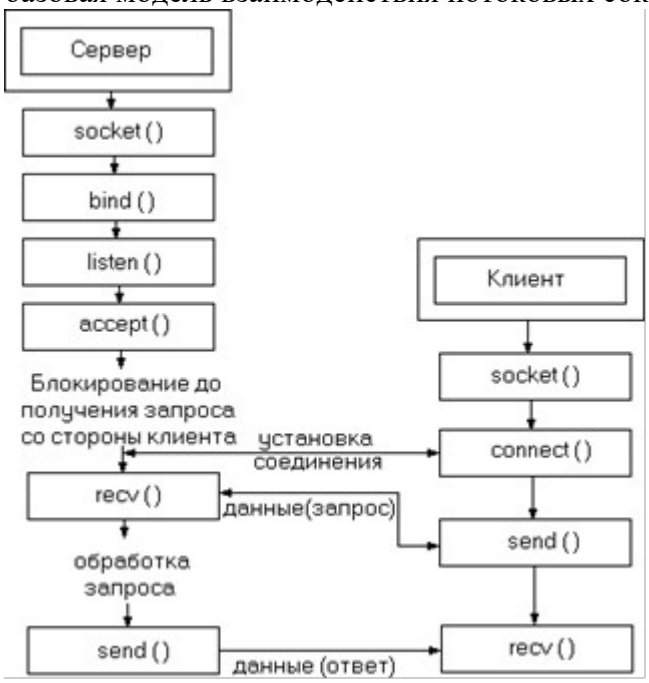


Рис. 1 Блок-схема взаимодействия потоковых сокетов

11. Создание сокета

Сокет идентифицирует пару, состоящую из IP-адреса и номера порта. Пара сокетов идентифицирует четыре компонента: адреса и номера портов отправителя и получателя.

Обращение к сокетам осуществляется при помощи соответствующих дескрипторов сокетов. В Win32 сокет отличается от описателя файла и представлен отдельным типом – SOCKET.

Сокет создается одной из двух функций:

```
SOCKET WSASocket (
        int af,
        int type,
        int protocol,
        LPWSAPROTOCOL_INFO IpProtocolInfo,
        GROUP g,
        DWORD dwFlags);
```

или SOCKET socket (int af, int type, int protocol);

Первый параметр – *af*, определяет семейство адресов протокола и ограничивает применение второго и третьего параметров. Он может принимать значения *AF_UNIX,*

AF_INET, AF_OSI, AF_NS, AF_NETBIOS, AF_APPLETALK, AF_ATM и т.д.. Значение *AF_INET* позволяет взаимодействовать через объединенную сеть по IP-адресам.

Параметр *type* - это тип сокета для данного протокола. Для протокола TCP/IP он может принимать одно из следующих значений: *SOCK_STREAM* (транспортный протокол TCP, ориентированная на соединение надежная связь), *SOCK_DGRAM* (транспортный протокол UDP, ненадежная дейтаграммная связь), *SOCK_RAW* (простые сокеты).

Третий параметр — *protocol,* указывает конкретный протокол. Если этот параметр равен 0, это означает, что задействуется протокол по умолчанию для выбранных значений семейства адресов и типа. Для протокола TCP задается значение *IPPROTO_IP*, для протокола UDP – *IPPROTO_UDP*, для простых сокетов – *IPPROTO_RAW* или *IPPROTO_ICMP*.

Если в функции WSASocket указать константу *FROM_PROTOCOL_INFO* во всех трех параметрах (*af, type* и *protocol)* — для них будут использоваться значения из переданной структуры *WSAPROTOCOL_INFO.*

Теперь рассмотрим два последних флага из *WSASocket.* Параметр *GROUP* всегда равен 0, так как ни одна из версий Winsock не поддерживает группы сокетов. В параметре *dwFlags* указывают один или несколько следующих флагов:

WSA_FLAG_OVERLAPPED;
WSA_FLAG_MULTIPOINT_C_ROOT;
WSA_FLAG_MULTIPOINT_C_LEAF;
WSA_FLAG_MULTIPOINT_D_ROOT;
WSA_FLAG_MULTIPOINT_D_LEAF.

Первый флаг — *WSA_FLAG_OVERLAPPED,* указывает, что данный сокет допускает перекрытый ввод-вывод. Если сокет создается функцией *socket,* флаг *WSA_FLAG_OVERLAPPED* задан по умолчанию. В общем, рекомендуется всегда задавать этот флаг при использовании *WSASocket.* Последние четыре флага относятся к сокетам многоадресного вещания.

В случае успеха функция *socket()* возвращает дескриптор сокета, представляющий собой целое число, в случае неудачи возвращается –1.

12. Проверка и обработка ошибок

Большинство функций Winsock при успешном завершении возвращают 0, а в случае ошибки - значение SOCKET_ERROR, которое равно –1.

Для получения кода ошибки можно использовать функцию int WSAGetLastError (void);

Функция возвращает код последней ошибки. Коды ошибок описаны в Winsock.h или Winsock2.h (в зависимости от версии)

13. Серверные API-функции

Первый шаг после создания сокета — привязка сокета данного протокола к его стандартному адресу функцией *bind.* Второй — перевод сокета в режим прослушивания функцией *listen.* И наконец, сервер должен принять соединение клиента функцией *accept* или *WSAAccept.*

Фукция bind.

```
int bind (
        SOCKET                  s,
        const struct  sockaddr FAR*  name,
        int                     namelen);
```

Параметр *s* задает дескриптор локального сокета, создаваемый функцией socket(), на котором ожидаются соединения клиентов.

Второй параметр – указатель на структуру, в которой хранится адрес сокета, соответствующий стандартам используемого протокола. Его нужно привести к типу *struct sockaddr.* В заголовочном файле Winsock определен тип *SOCKADDR,* соответствующий структуре *struct sockaddr.*

Третий параметр задает размер переданной адресной структуры, зависящий от протокола.

В случае успеха функция *bind()* возвращает 0. В случае ошибки - значение *SOCKET_ERROR* (т.е. -1). Самая распространенная ошибка при вызове *bind* — *WSAEADDRINUSE.* В случае использования TCP/IP это означает, что с локальным IP-интерфейсом и номером порта уже связан другой процесс, или они находятся в состоянии *TIME_WAIT.* При повторном вызове *bind* для уже связанного сокета возвращается ошибка *WSAEFAULT.*

Функция listen.

Для перевода сокета в состояние ожидания входящих соединений используется API-функция *listen().*

int listen(SOCKET s, int backlog);

Первый параметр *s* — дескриптор сокета.

Параметр *backlog* – максимальное число входящих запросов на установление соединения, которые могут ждать в очереди на обработку пока сервер занят. Значение *backlog* зависит от поставщика протокола. Недопустимое значение заменяется ближайшим разрешенным.

Функции accept.

Прототип функции *accept:*

```
SOCKET accept(
        SOCKET s,
        struct sockaddr FAR* addr,
        int FAR* addrlen );
```

Параметр *s* — связанный сокет в состоянии прослушивания.

Второй параметр — адрес действительной структуры *SOCKADDR_IN.*

Параметр *addrlen* — ссылка на длину структуры *SOCKADDR_IN.*

Вызов *accept* обслуживает первый находящийся в очереди запрос на соединение. По его завершении структура *addr* будет содержать сведения об IP-адресе клиента, отправившего запрос, а параметр *addrlen* — размер структуры.

Кроме того, *accept* возвращает новый дескриптор сокета, соответствующий принятому клиентскому соединению. Для всех последующих операций с этим клиентом должен применяться новый сокет. Исходный прослушивающий сокет используется для приема других клиентских соединений и продолжает находиться в режиме прослушивания.

В Winsock 2 есть функция *WSAAccept,* способная устанавливать соединения в зависимости от результата вычисления условия.

14. Клиентские API-функции

Установление соединения осуществляется вызовом *connect* или *WSAConnect.*

Прототип функции connect:

```
int connect (
        SOCKET s,
        const struct sockaddr FAR* name,
        int namelen);
```

Параметр *s* — действительный TCP-сокет для установления соединения, *name* — структура адреса сокета *(SOCKADDR_IN)* для TCP, описывающая сервер к которому подключаются, *namelen* — длина переменной *name.*

Если на компьютере, к которому вы подключаетесь, не запущен процесс, прослушивающий данный порт, функция *connect* вернет ошибку *WSAECONNREFUSED.* Другая ошибка — *WSAETIMEDOUT,* происходит, когда вызываемый адресат недоступен, например, из-за отказа коммуникационного оборудования на пути к узлу или отсутствия узла в сети.

15. Передача и прием данных по сокету (потоковые протоколы)

Для пересылки данных по сокету используются функции *send* и *WSASend.* Аналогично, для приема данных существуют функции *recv* и *WSARecv.*

Все буферы, используемые при отправке и приеме данных, состоят из элементов типа *char.*

Функция send.

API-функция *send* для отправки данных по сокету определена так:

```
int send(
    SOCKET s,
    const char FAR * buf,
    int len,
    int flags );
```

Параметр s определяет сокет для отправки данных.

Второй параметр — *buf,* указывает на символьный буфер, содержащий данные для отправки.

Третий — *len,* задает число отправляемых из буфера символов.

Параметр — *flags,* может принимать значения 0, *MSG_DONTROUTE, MSG_OOB,* или результат логического ИЛИ над любыми из этих параметров. При указании флага *MSG_DONTROUTE* не будут маршрутизироваться отправляемые пакеты. Флаг *MSG_OOB* указывает, что данные должны быть отправлены *вне полосы* (out of band), то есть срочно.

При успешном выполнении функция *send* вернет количество переданных байт, иначе — ошибку *SOCKET_ERROR.* Число, возвращаемое функцией *send* может быть меньше указанного размера буфера. Одна из типичных ошибок — *WSAE-CONNABORTED,* происходит при разрыве виртуального соединения из-за ошибки протокола или истечения времени ожидания. В этом случае сокет должен быть закрыт, так как он больше не может использоваться.

Число, возвращаемое функцией *send* может быть меньше указанного размера буфера.

В Winsock версии 2 определена функция *WSASend* — аналог *send.*

Функция recv.

Функция *recv* определена так:

```
int recv(
    SOCKET s,
    char FAR * buf,
    int len,
    int flags );
```

Параметр s определяет сокет для приема данных.

Второй параметр — *buf,* является символьным буфером и предназначен для полученных данных.

Параметр *len* указывает число принимаемых байт или размер буфера *buf.*

Последний параметр — *flags,* может принимать значения 0, *MSG_PEEK, MSG_OOB* или результат логического ИЛИ над любыми из этих параметров. Разумеется, 0 означает отсутствие особых действий. Флаг *MSG_PEEK* указывает, что доступные данные должны копироваться в принимающий буфер и при этом оставаться в системном буфере. Его использовать не рекомендуется. Флаг *MSG_OOB* уже обсуждался при рассмотрении отправки данных.

Использование *recv* в сокетах, ориентированных на передачу сообщений или дейтаграмм, имеет несколько особенностей. Если при вызове *recv* размер ожидающих обработки данных больше предоставляемого буфера, то после его полного заполнения возникает ошибка *WSAEMSGSIZE.* Заметьте: ошибка превышения размера сообщения происходит только при использовании протоколов, ориентированных на передачу сообщений. Потоковые протоколы буферизируют поступающие данные и при запросе приложением предоставляют их в полном объеме, даже если количество ожидающих

обработки данных больше размера буфера. Таким образом, ошибка *WSAEMSGSIZE* не может произойти при работе с потоковыми протоколами.

Функция *WSARecv* обладает дополнительными по сравнению с *recv* возможностями: поддерживает перекрытый ввод-вывод и фрагментарные дейтаграммные уведомления.

16. Завершение сеанса

По окончании работы с сокетом необходимо закрыть соединение и освободить все ресурсы, связанные с описателем сокета, вызвав функцию *closesocket.* Ее неправильное использование может привести к потере данных. Поэтому перед вызовом *closesocket* сеанс нужно корректно завершить функцией *shutdown.*

Функция shutdown.

Корректное завершение сеанса, при котором приложение уведомляет получателя об окончании отправки данных, осуществляется с помощью функции *shutdown.*

```
int shutdown(
    SOCKET s,
    int how );
```

Параметр *how* может принимать значения *SD_IRECEIVE, SD_SEND* или *SD_ BOTH.* Значение *SD_IRECEIVE* запрещает все последующие вызовы любых функций приема данных, на протоколы нижнего уровня это не действует. Если в очереди TCP-сокета есть данные, либо они поступают позже, соединение сбрасывается. UDP-сокеты в аналогиной ситуации продолжают принимать данные и ставить их в очередь.

SD_SEND запрещает все последующие вызовы функций отправки данных. В случае TCP-сокетов после подтверждения получателем приема всех отправленных данных передается пакет FIN.

SD_BOTH запрещает как прием, так и отправку.

Функция closesocket.

Эта функция закрывает сокет. Она определена так: int closesocket (SOCKET s);

Вызов *closesocket* освобождает дескриптор сокета, и все дальнейшие операции с сокетом закончатся ошибкой *WSAENOTSOCK.* Если не существует других ссылок на сокет, все связанные с дескриптором ресурсы будут освобождены, включая данные в очереди.

17. Протоколы без предварительного установления соединения

При использовании дейтаграммных сокетов сначала создают сокет функцией *socket* или *WSASocket.* Затем выполняют привязку сокета к интерфейсу, на котором будут принимать данные, функцией *bind* (как и в случае протоколов, ориентированных на сеансы). Разница в том, что вместо вызова *listen* или *accept* нужно просто ожидать приема входящих данных. Поскольку в этом случае соединения нет, принимающий сокет может получать дейтаграммы от любой машины в сети. Простейшая функция приема — *recvform:*

```
int recvfrom(
    SOCKET s,
    char FAR* buf,
    int len,
    int flags,
    struct sockaddr FAR* from,
    int FAR* fromlen );
```

Первые четыре параметра такие же, как и для функции *recv.* Параметр *from* — структура *SOCKADDR* для данного протокола слушающего сокета, на размер структуры адреса ссылается *fromlen.* После возврата вызова структура *SOCKADDR* будет содержать адрес рабочей станции, которая отправляет данные.

В Winsock 2 применяется другая версия *recvform — WSARecvForm.*

Другой способ приема (отправки) данных в сокетах, не требующих соединения, — установление соединения (хоть это и звучит странно). После создания сокета можно вызвать *connect или WSAConnect,* присвоив параметру *SOCKADDR* адрес удаленного компьютера, с которым необходимо связаться. Фактически никакого соединения не происходит. Адрес сокета, переданный в функцию соединения, ассоциируется с сокетом, чтобы было можно использовать функции *recv* и *WSARecv* вместо *recvfrom* или *WSARecvFrom* (поскольку источник данных известен).

Есть два способа отправки данных через сокет, не требующий соединения. Первый и самый простой — создать сокет и вызвать функцию *sendto* или *WSASendTo.* Рассмотрим функцию *sendto:*

```
int sendto(
    SOCKET s,
    const char FAR * buf,
    int len,
    int flags,
    const struct sockaddr FAR * to,
    int tolen );
```

Параметры этой функции такие же, как и у *recvfrom,* за исключением *buf*— буфера данных для отправки, и *len* — показывающего сколько байт отправлять. Параметр *to* — указатель на структуру *SOCKADDR* с адресом принимающей рабочей станции.

Также можно использовать функцию *WSASendTo* из Winsock 2:

Как и при получении данных, сокет, не требующий соединения, можно подключать к адресу конечной точки и отправлять данные функциями *send* и *WSASend.* После создания этой привязки использовать для обмена данными функции *sendto* или *WSASendTo с* другим адресом нельзя — будет выдана ошибка. Отменить привязку сокета можно, лишь вызвав функцию *closesocket* с описателем этого сокета, после чего следует создать новый сокет.

Поскольку соединение не устанавливается, его формального разрыва или корректного закрытия не требуется. После прекращения отправки или получения данных отправителем или получателем просто вызывается функция *closesocket* с описателем требуемого сокета, в результате чего освобождаются все выделенные ему ресурсы.

10. **Порядок выполнения работы**

Задание 1.

Разработать TCP-сервер, создающий сокет, привязывающий его к локальному IP- адресу и порту и прослушивающий соединения клиентов. Номер порта и IP-адрес вводить с клавиатуры. IP-адрес задавать в десятично-точечной нотации.

Учесть, что функция accept возвращает новый дескриптор сокета, соответствующий принятому клиентскому соединению. Для всех последующих операций с данным клиентским соединением должен применяться новый сокет. Исходный прослушивающий сокет используется для приема других клиентских соединений и продолжает находиться в режиме прослушивания. При получении от клиента запроса на установление соединения вывести на экран IP-адрес и номер порта клиентского сокета.

Разработать приложение–клиент, соединяющееся с заданным TCP-сервером. Все отправленные и полученные по сокету данные вывести на экран.

При вызове API-функций выполнять проверку и обработку ошибок.

Нарисовать блок-схему серверной и клиентской части программы.

Задание 2.

Разработать серверное приложение, выполняющее получение данных через сокет без установления соединения по протоколу UDP.

Разработать клиентское приложение, выполняющее отправку UDP-дейтаграмм. IP-адрес и порт удаленного получателя должен задаваться пользователем.

При вызове API-функций выполнять проверку и обработку ошибок.

Адреса сокетов вывести на экран.

Нарисовать блок-схему серверной и клиентской части программы.

11. Варианты заданий:

1) Удаленный калькулятор (+,-,*,/);
2) Работа с массивом чисел (количество элементов в массиве, получение значения элемента массива по номеру, найти номер элемента в массиве по значению, увеличить значения элементов на заданное число);
3) Работа с массивом символов (преобразовать символы в верхний регистр, в нижний регистр, проверить, является ли символ цифрой, буквой, получить значение элемента массива);
4) Удаленный однострочный редактор (вставка символа в позицию, в конец строки, удаление символа из позиции);
5) Редактор для работы с двумя строками (конкатенация строк, проверка на равенство строк, проверка на равенство длин, вставка второй строки в заданную позицию первой);
6) Удаленный генератор псевдослучайных последовательностей (одно целое число в диапазоне, массив чисел в диапазоне).

При разработке программ использовать функции Windows API или класс Socket пространства имен System.Net .NET Framework. В качестве языков программирования - C++, либо C#.

12. Контрольные вопросы

1. Что такое сокет? Какая версия Winsock используется на платформе Win32?
2. Сколько сокетов необходимо для взаимодействия клиента и сервера? Что представляет собой адрес сокета?
3. Назовите типы сокетов. В каком случае предпочтительнее использовать тот или иной тип сокетов?
4. Какой адрес можно использовать, чтобы прослушивать все сетевые интерфейсы локального узла?
5. Какой порядок байтов используется в Intel-совместимых процессорах? Какой порядок байтов применяется для работы с сокетами?
6. Какие номера портов могут задействовать прикладные программы при работе с сокетами?
7. Какая функция позволяет узнать номера портов, используемых стандартными службами? В каком файле хранится эта информация?
8. Чем отличается процесс получения и отправки данных на сокете, не требующем соединения?
9. Как осуществляется корректное завершение сеанса работы с сокетом? В чем отличие завершения работы с потоковым и дейтаграммным сокетом?

Практическая работа №4 Работы с буфером экрана

Для выполнения практической работы потребуется отладочная плата 1986EvBrd_48_Rev3. В этой практической работе дан пример реализации драйвера дисплея и вывода текста на русском языке на экран, а так же организация вывода счётчика.

Проект состоит из восьми файлов. Один из них написан на языке ассемблер, остальные на языке си.

Описание файлов

В файле font_ansi_5x8.c представлен шрифт в кодировке ANSI. Каждый из символов представлен в виде 5 байт (5 на 8 бит). Благодаря последовательности расположения описания символов (в том числе букв) в порядке следования определённом в кодировке ANSI имеется возможность в программе писать тексты на русском языке в удобочитаемом виде.

В файле graphics.c размещены функции для вывода текста в буфер, очистки буфера и выгрузки буфера на дисплей.

Файл startup_1986be9x.s

```
1           AREA    RESET, DATA, READONLY
2           IMPORT              main
3           DCD     0x20008000      ; Вершина стека
4           DCD     main            ; Выполнение программы main
5           END
```

Файл structs.h

```
#ifndef STRUCTS
#define STRUCTS
3
/* Определение структуры порта */
typedef struct {
unsigned RXTX;
unsigned OE;
unsigned FUNC;
unsigned DIGITAL;
unsigned PULL;
unsigned PD;
unsigned PWR;
unsigned GFEN;
14}struct_ports;
15
16#define A ((struct_ports *) 0x400A8000)
17#define D ((struct_ports *) 0x400C0000)
18#define F ((struct_ports *) 0x400E8000)
19
20#endif
```

Файл driver_lcd.h

```
1  #ifndef DRIVER_LCD
2  #define DRIVER_LCD
3
4  #include "structs.h"
5
6  void LcdInit(void);
7  void LcdDispOn(void);
```

```
8 void LcdScreen(unsigned char screen[8][128]);
9
10#endif //DRIVER_LCD
```

Файл font_ansi_5x8.h

```
1#ifndef FONT_ANSI_5X8
2#define FONT_ANSI_5X8
3
4extern const unsigned char ansi[256][5];
5
6#endif //FONT_ANSI_5X8
```

Файл graphics.h

```
1#include "structs.h"
2#include "driver_lcd.h"
3#include "font_ansi_5x8.h"
4
5void Print(unsigned row, unsigned col, char * byte);
6void ClearScreen(void);
7void ScreenUpdate(void);
```

Файл main.c

```
1  /*                                        */
2  /* Практическая работа 02 (буфер дисплея) */
3  /*                                        */
4
#include "structs.h"
#include "driver_lcd.h"
#include "graphics.h"
8
9  void PortOut(struct_ports * port){
10    port->OE      = 0xFF;      /* Выход              */
11    port->FUNC    = 0x0000;    /* Порт               */
12    port->DIGITAL = 0xFF;      /* Цифровые           */
13    port->PULL    = 0x00000000; /* Подтяжка отключена   */
14    port->PD      = 0x00000000; /* Управляемый драйвер  */
15    port->PWR     =                 /* Фронт ~10 нс       */
16    0xFFFF; port->GFEN          /* Фильтрация отключена */
17}                       = 0x0000;
18
19static void Delay(unsigned count){
20  while(count--);
21}
22
23int main (void) {
24    /* Включение тактирования порта F */
25    *(unsigned *)(0x4002001C) |= (1<<29);
26    PortOut(F);
27
28    /* Включение тактирования порта D */
29    *(unsigned *)(0x4002001C) |= (1<<24);
```

```
30  PortOut(D);
31
32  /* Включение тактирования порта A */
33  *(unsigned *)(0x4002001C) |= (1<<21);
34  PortOut(A);
35
36  LcdInit();
37  LcdDispOn();
38  ClearScreen();
39
40  Print(0,5,"Привет, программист!");
41  Print(2,0,"Это пример вывода    ");
42
43  Print(3,0,"текста на LCD-дисплей");
44
45  ScreenUpdate();
46  Print(7,0,"Счет:");
47  char str[2] = "00";
48  for(int i = 35; i >= 0; --i){
49     str[0] = '0' + i/10;
50     str[1] = '0' + i%10;
51     Print(7,36,str);
52     ScreenUpdate();
53     Delay(2000000);
54  }
55
56  ClearScreen();
57  Print(3,25,"Нажмите RESET");
58  Print(4,19,"для перезапуска");
59  ScreenUpdate();
60
61  return 0;
62}
```

Файл driver_lcd.c

```
1   #include "driver_lcd.h"
2
3   /* Обозначения в соответствии с описанием дисплея */
4   /* MT-12864Jv.1 МЭЛТ */
5   #define LCD_SET_COL 0x40
6   #define LCD_SET_ROW 0xB8
7
8   #define Pause /*Delay(5)*/
9   #define Pause2ms Delay(100)
10
11  #define LCD_MT_DB A->RXTX
12  #define LCD_MT_RW_off F->RXTX &= ~(1<<3)
13  #define LCD_MT_RW_on F->RXTX |= (1<<3)
14  #define LCD_MT_E
15  #define LCD_MT_RES
16  #define LCD_MT_A0_on F->RXTX |= (1<<4)
17  #define LCD_MT_A0_off F->RXTX &= ~(1<<4)
```

```
18 #define LCD_MT_E1 F->RXTX |= (F->RXTX & ~0x3) | 0x1
19 #define LCD_MT_E2 F->RXTX |= (F->RXTX & ~0x3) | 0x2
20 #define LCD_MT_E3 F->RXTX |= 0x3
21
22 /* Реализация функции задержки времени */
23 static void Delay(unsigned count){
24 while(count--);
25 }
26
27 /* Строб для записи байта в дисплей */
28 static void Strob(void){
29    Pause; /* Необходимая пауза */
30    D->RXTX |=  (1<<3); /* Включаем  3 бит порта D */ // Set_Stb_Pin
31    Pause; /* Необходимая пауза */
32    D->RXTX &= ~(1<<3); /* Выключаем 3 бит порта D */ // Set_Stb_Pin
33    Pause; /* Необходимая пауза */
34 }
35
36 /* Инициализация дисплея */
37 void LcdInit(void){
38    F->RXTX &= ~(1<<2); /* Выключаем второй бит порта F */ // Clr_Res
39    Pause2ms; /* Необходимая пауза */
40    F->RXTX |= (1<<2); /* Включаем 2 бит порта F */ // Set_Res
41    LCD_MT_E3; /* Включаем 0 и 1 бит порта F */ // Set_E1_Pin
42    LCD_MT_A0_off;
43    LCD_MT_RW_on;
44    Strob();
45    LCD_MT_A0_off;
46    LCD_MT_RW_off;
47 }
48
49 /* Включение дисплея */
50 void LcdDispOn(void){
51     LCD_MT_A0_off;
52    LCD_MT_RW_off;
53    Pause; /* Необходимая пауза */
54    A->RXTX |= 0x3F; /* Младшие 6 бит в 1 */
55    Strob();
56 }
57
58 /* Выбор чипа */
59 static void SetChip(unsigned num_chip){
60    F->RXTX &= ~(3<<0);
61    F->RXTX |=  (1<<num_chip);
62 }
63
64 /* Выбор ряда */
65 static void SetRow(unsigned char row)
66    { LCD_MT_A0_off;
67    LCD_MT_RW_off;
68    A->RXTX = (LCD_SET_ROW | row);
69    Strob();
```

```
70  }
71
/* Выбор колонки */
static void SetCol(unsigned char col){
LCD_MT_A0_off;
LCD_MT_RW_off;
A->RXTX = (LCD_SET_COL | col);
Strob(); 78  }
79
/* Вывод байта */
static void LcdWriteData(unsigned char data){
LCD_MT_A0_on;
LCD_MT_RW_off;
A->RXTX = data;
Strob();
LCD_MT_A0_off;
LCD_MT_RW_off; 88  }
89
/* Вывод изображения из буфера на экран */
void LcdScreen(unsigned char screen[8][128]){
SetChip(0);
for(unsigned row = 0; row < 8; ++row){
SetRow(row);
SetCol(0);
for(unsigned col = 0; col < 64; ++col){
unsigned char byte = screen[row][col];
LcdWriteData(byte); 99          }
100    }
SetChip(1);
for(unsigned row = 0; row < 8; ++row){
SetRow(row);
SetCol(0);
for(unsigned col = 64; col < 128; ++col){
unsigned char byte = screen[row][col];
LcdWriteData(byte);
108        }
109    }
110}
```

Файл font_ansi_5x8.c

```
1   #include "font_ansi_5x8.h"
2
3   const unsigned char ansi[256][5] = {
4       {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x00
5       {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x01
6       {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x02
7       {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x03
8       {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x04
9       {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x05
```

```
10      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x06
11      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x07
12      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x08
13      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x09
14      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0A
15      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0B
16      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0C
17      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0D
18      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0E
19      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0F
20
21      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x10
22      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x11
23      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x12
24      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x13
25      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x14
26      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x15
27      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x16
28      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x17
29      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x18
30      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x19
31      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1A
32      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1B
33      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1C
34      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1D
35      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1E
36      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1F
37
38      {0x00, 0x00, 0x00, 0x00, 0x00}, // sp 0x20
39      {0x00, 0x00, 0x5f, 0x00, 0x00}, // ! 0x21
40      {0x00, 0x07, 0x00, 0x07, 0x00}, // " 0x22
41      {0x14, 0x7f, 0x14, 0x7f, 0x14}, // # 0x23
42      {0x24, 0x2a, 0x7f, 0x2a, 0x12}, // $ 0x24
43      {0x23, 0x13, 0x08, 0x64, 0x62}, // % 0x25
44      {0x36, 0x49, 0x55, 0x22, 0x50}, // & 0x26
45      {0x00, 0x05, 0x03, 0x00, 0x00}, // ' 0x27
46      {0x00, 0x1c, 0x22, 0x41, 0x00}, // ( 0x28
47      {0x00, 0x41, 0x22, 0x1c, 0x00}, // ) 0x29
48      {0x14, 0x08, 0x3E, 0x08, 0x14}, // * 0x2A
49      {0x08, 0x08, 0x3E, 0x08, 0x08}, // + 0x2B
50      {0x00, 0x00, 0x50, 0x30, 0x00}, // , 0x2C
51      {0x10, 0x10, 0x10, 0x10, 0x10}, // - 0x2D
52      {0x00, 0x60, 0x60, 0x00, 0x00}, // . 0x2E
53      {0x20, 0x10, 0x08, 0x04, 0x02}, // / 0x2F
54      {0x3E, 0x51, 0x49, 0x45, 0x3E}, // 0 0x30
55      {0x00, 0x42, 0x7F, 0x40, 0x00}, // 1 0x31
56      {0x42, 0x61, 0x51, 0x49, 0x46}, // 2 0x32
57      {0x21, 0x41, 0x45, 0x4B, 0x31}, // 3 0x33
58      {0x18, 0x14, 0x12, 0x7F, 0x10}, // 4 0x34
59      {0x27, 0x45, 0x45, 0x45, 0x39}, // 5 0x35
60      {0x3C, 0x4A, 0x49, 0x49, 0x30}, // 6 0x36
61      {0x01, 0x71, 0x09, 0x05, 0x03}, // 7 0x37
```

```
62    {0x36, 0x49, 0x49, 0x49, 0x36}, // 8 0x38
63    {0x06, 0x49, 0x49, 0x29, 0x1E}, // 9 0x39
64    {0x00, 0x36, 0x36, 0x00, 0x00}, // : 0x3A
65    {0x00, 0x56, 0x36, 0x00, 0x00}, // ; 0x3B
66    {0x08, 0x14, 0x22, 0x41, 0x00}, // < 0x3C
67    {0x14, 0x14, 0x14, 0x14, 0x14}, // = 0x3D
68    {0x00, 0x41, 0x22, 0x14, 0x08}, // > 0x3E
69    {0x02, 0x01, 0x51, 0x09, 0x06}, // ? 0x3F
70    {0x32, 0x49, 0x59, 0x51, 0x3E}, // @ 0x40
71    {0x7E, 0x11, 0x11, 0x11, 0x7E}, // A 0x41
72    {0x7F, 0x49, 0x49, 0x49, 0x36}, // B 0x42
73    {0x3E, 0x41, 0x41, 0x41, 0x22}, // C 0x43
74    {0x7F, 0x41, 0x41, 0x22, 0x1C}, // D 0x44
75    {0x7F, 0x49, 0x49, 0x49, 0x41}, // E 0x45
76    {0x7F, 0x09, 0x09, 0x09, 0x01}, // F 0x46
77    {0x3E, 0x41, 0x49, 0x49, 0x7A}, // G 0x47
78    {0x7F, 0x08, 0x08, 0x08, 0x7F}, // H 0x48
79    {0x00, 0x41, 0x7F, 0x41, 0x00}, // I 0x49
80    {0x20, 0x40, 0x41, 0x3F, 0x01}, // J 0x4A
81    {0x7F, 0x08, 0x14, 0x22, 0x41}, // K 0x4B
82    {0x7F, 0x40, 0x40, 0x40, 0x40}, // L 0x4C
83    {0x7F, 0x02, 0x0C, 0x02, 0x7F}, // M 0x4D
84    {0x7F, 0x04, 0x08, 0x10, 0x7F}, // N 0x4E
85    {0x3E, 0x41, 0x41, 0x41, 0x3E}, // O 0x4F
86    {0x7F, 0x09, 0x09, 0x09, 0x06}, // P 0x50
87    {0x3E, 0x41, 0x51, 0x21, 0x5E}, // Q 0x51
88    {0x7F, 0x09, 0x19, 0x29, 0x46}, // R 0x52
89    {0x46, 0x49, 0x49, 0x49, 0x31}, // S 0x53
90    {0x01, 0x01, 0x7F, 0x01, 0x01}, // T 0x54
91    {0x3F, 0x40, 0x40, 0x40, 0x3F}, // U 0x55
92    {0x1F, 0x20, 0x40, 0x20, 0x1F}, // V 0x56
93    {0x3F, 0x40, 0x38, 0x40, 0x3F}, // W 0x57
94    {0x63, 0x14, 0x08, 0x14, 0x63}, // X 0x58
95    {0x07, 0x08, 0x70, 0x08, 0x07}, // Y 0x59
96    {0x61, 0x51, 0x49, 0x45, 0x43}, // Z 0x5A
97    {0x00, 0x7F, 0x41, 0x41, 0x00}, // [ 0x5B
98    {0x55, 0x2A, 0x55, 0x2A, 0x55}, // 0x5C
99    {0x00, 0x41, 0x41, 0x7F, 0x00}, // ] 0x5D
100   {0x04, 0x02, 0x01, 0x02, 0x04}, // ^ 0x5E
101   {0x40, 0x40, 0x40, 0x40, 0x40}, // _ 0x5F
102   {0x00, 0x01, 0x02, 0x04, 0x00}, // ' 0x60
103   {0x20, 0x54, 0x54, 0x54, 0x78}, // a 0x61
104   {0x7F, 0x48, 0x44, 0x44, 0x38}, // b 0x62
105   {0x38, 0x44, 0x44, 0x44, 0x20}, // c 0x63
106   {0x38, 0x44, 0x44, 0x48, 0x7F}, // d 0x64
107   {0x38, 0x54, 0x54, 0x54, 0x18}, // e 0x65
108   {0x08, 0x7E, 0x09, 0x01, 0x02}, // f 0x66
109   {0x0C, 0x52, 0x52, 0x52, 0x3E}, // g 0x67
110   {0x7F, 0x08, 0x04, 0x04, 0x78}, // h 0x68
111   {0x00, 0x44, 0x7D, 0x40, 0x00}, // i 0x69
112   {0x20, 0x40, 0x44, 0x3D, 0x00}, // j 0x6A
113   {0x7F, 0x10, 0x28, 0x44, 0x00}, // k 0x6B
```

```
114   {0x00, 0x41, 0x7F, 0x40, 0x00}, // l 0x6C
115   {0x7C, 0x04, 0x18, 0x04, 0x78}, // m 0x6D
116   {0x7C, 0x08, 0x04, 0x04, 0x78}, // n 0x6E
117   {0x38, 0x44, 0x44, 0x44, 0x38}, // o 0x6F
118   {0x7C, 0x14, 0x14, 0x14, 0x08}, // p 0x70
119   {0x08, 0x14, 0x14, 0x18, 0x7C}, // q 0x71
120   {0x7C, 0x08, 0x04, 0x04, 0x08}, // r 0x72
121   {0x48, 0x54, 0x54, 0x54, 0x20}, // s 0x73
122   {0x04, 0x3F, 0x44, 0x40, 0x20}, // t 0x74
123   {0x3C, 0x40, 0x40, 0x20, 0x7C}, // u 0x75
124   {0x1C, 0x20, 0x40, 0x20, 0x1C}, // v 0x76
125   {0x3C, 0x40, 0x30, 0x40, 0x3C}, // w 0x77
126   {0x44, 0x28, 0x10, 0x28, 0x44}, // x 0x78
127   {0x0C, 0x50, 0x50, 0x50, 0x3C}, // y 0x79
128   {0x44, 0x64, 0x54, 0x4C, 0x44}, // z 0x7A
129   {0x08, 0x08, 0x36, 0x41, 0x41}, // { 0x7B
130   {0x00, 0x00, 0x7F, 0x00, 0x00}, // | 0x7C
131   {0x41, 0x41, 0x36, 0x08, 0x08}, // } 0x7D
132   {0x02, 0x01, 0x02, 0x02, 0x01}, // ~ 0x7E
133   {0x00, 0x00, 0x00, 0x00, 0x00}, // DE 0x7F
134
135   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x80
136   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x81
137   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x82
138   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x83
139   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x84
140   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x85
141   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x86
142   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x87
143   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x88
144   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x89
145   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8A
146   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8B
147   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8C
148   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8D
149   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8E
150   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8F
151
152   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x90
153   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x91
154   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x92
155   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x93
156   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x94
157   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x95
158   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x96
159   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x97
160   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x98
161   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x99
162   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9A
163   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9B
164   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9C
165   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9D
```

```
166   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9E
167   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9F
168
169   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA0
170   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA1
171   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA2
172   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA3
173   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA4
174   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA5
175   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA6
176   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA7
177   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA8
178   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA9
179   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAA
180   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAB
181   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAC
182   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAD
183   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAE
184   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAF
185
186   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB0
187   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB1
188   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB2
189   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB3
190   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB4
191   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB5
192   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB6
193   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB7
194   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB8
195   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB9
196   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBA
197   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBB
198   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBC
199   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBD
200   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBE
201   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBF
202
203   {0x7E, 0x11, 0x11, 0x11, 0x7E}, // А 0xC0
204   {0x7F, 0x49, 0x49, 0x49, 0x33}, // Б 0xC1
205   {0x7F, 0x49, 0x49, 0x49, 0x36}, // В 0xC2
206   {0x7F, 0x01, 0x01, 0x01, 0x03}, // Г 0xC3
207   {0xE0, 0x51, 0x4F, 0x41, 0xFF}, // Д 0xC4
208   {0x7F, 0x49, 0x49, 0x49, 0x49}, // Е 0xC5
209   {0x77, 0x08, 0x7F, 0x08, 0x77}, // Ж 0xC6
210   {0x49, 0x49, 0x49, 0x49, 0x36}, // З 0xC7
211   {0x7F, 0x10, 0x08, 0x04, 0x7F}, // И 0xC8
212   {0x7C, 0x21, 0x12, 0x09, 0x7C}, // Й 0xC9
213   {0x7F, 0x08, 0x14, 0x22, 0x41}, // К 0xCA
214   {0x20, 0x41, 0x3F, 0x01, 0x7F}, // Л 0xCB
215   {0x7F, 0x02, 0x0C, 0x02, 0x7F}, // М 0xCC
216   {0x7F, 0x08, 0x08, 0x08, 0x7F}, // Н 0xCD
217   {0x3E, 0x41, 0x41, 0x41, 0x3E}, // О 0xCE
```

```
218   {0x7F, 0x01, 0x01, 0x01, 0x7F}, // П 0xCF
219   {0x7F, 0x09, 0x09, 0x09, 0x06}, // Р 0xD0
220   {0x3E, 0x41, 0x41, 0x41, 0x22}, // С 0xD1
221   {0x01, 0x01, 0x7F, 0x01, 0x01}, // Т 0xD2
222   {0x27, 0x48, 0x48, 0x48, 0x3F}, // У 0xD3
223   {0x1C, 0x22, 0x7F, 0x22, 0x1C}, // Ф 0xD4
224   {0x63, 0x14, 0x08, 0x14, 0x63}, // Х 0xD5
225   {0x7F, 0x40, 0x40, 0x40, 0xFF}, // Ц 0xD6
226   {0x07, 0x08, 0x08, 0x08, 0x7F}, // Ч 0xD7
227   {0x7F, 0x40, 0x7F, 0x40, 0x7F}, // Ш 0xD8
228   {0x7F, 0x40, 0x7F, 0x40, 0xFF}, // Щ 0xD9
229   {0x01, 0x7F, 0x48, 0x48, 0x30}, // Ъ 0xDA
230   {0x7F, 0x48, 0x30, 0x00, 0x7F}, // Ы 0xDB
231   {0x7F, 0x48, 0x48, 0x30, 0x00}, // Ь 0xDC
232   {0x22, 0x41, 0x49, 0x49, 0x3E}, // Э 0xDD
233   {0x7F, 0x08, 0x3E, 0x41, 0x3E}, // Ю 0xDE
234   {0x46, 0x29, 0x19, 0x09, 0x7F}, // Я 0xDF
235   {0x20, 0x54, 0x54, 0x54, 0x78}, // а 0xE0
236   {0x3C, 0x4A, 0x4A, 0x49, 0x31}, // б 0xE1
237   {0x7C, 0x54, 0x54, 0x28, 0x00}, // в 0xE2
238   {0x7C, 0x04, 0x04, 0x04, 0x0C}, // г 0xE3
239   {0xE0, 0x54, 0x4C, 0x44, 0xFC}, // д 0xE4
240   {0x38, 0x54, 0x54, 0x54, 0x08}, // е 0xE5
241   {0x6C, 0x10, 0x7C, 0x10, 0x6C}, // ж 0xE6
242   {0x44, 0x44, 0x54, 0x54, 0x28}, // з 0xE7
243   {0x7C, 0x20, 0x10, 0x08, 0x7C}, // и 0xE8
244   {0x78, 0x42, 0x24, 0x12, 0x78}, // й 0xE9
245   {0x7C, 0x10, 0x28, 0x44, 0x00}, // к 0xEA
246   {0x20, 0x44, 0x3C, 0x04, 0x7C}, // л 0xEB
247   {0x7C, 0x08, 0x10, 0x08, 0x7C}, // м 0xEC
248   {0x7C, 0x10, 0x10, 0x10, 0x7C}, // н 0xED
249   {0x38, 0x44, 0x44, 0x44, 0x38}, // о 0xEE
250   {0x7C, 0x04, 0x04, 0x04, 0x7C}, // п 0xEF
251   {0x7C, 0x14, 0x14, 0x14, 0x08}, // р 0xE0
252   {0x38, 0x44, 0x44, 0x44, 0x44}, // с 0xF1
253   {0x04, 0x04, 0x7C, 0x04, 0x04}, // т 0xF2
254   {0x0C, 0x50, 0x50, 0x50, 0x3C}, // у 0xF3
255   {0x18, 0x24, 0x7E, 0x24, 0x18}, // ф 0xF4
256   {0x44, 0x28, 0x10, 0x28, 0x44}, // х 0xF5
257   {0x7C, 0x40, 0x40, 0x40, 0xFC}, // ц 0xF6
258   {0x0C, 0x10, 0x10, 0x10, 0x7C}, // ч 0xF7
259   {0x7C, 0x40, 0x7C, 0x40, 0x7C}, // ш 0xF8
260   {0x7C, 0x40, 0x7C, 0x40, 0xFC}, // щ 0xF9
261   {0x04, 0x7C, 0x50, 0x50, 0x20}, // ъ 0xFA
262   {0x7C, 0x50, 0x20, 0x00, 0x7C}, // ы 0xFB
263   {0x7C, 0x50, 0x50, 0x20, 0x00}, // ь 0xFC
264   {0x28, 0x44, 0x54, 0x54, 0x38}, // э 0xFD
265   {0x7C, 0x10, 0x38, 0x44, 0x38}, // ю 0xFE
266   {0x08, 0x54, 0x34, 0x14, 0x7C}  // я 0xFF
267};
```

Файл graphics.c

```
1  #include "graphics.h"
2
3  /* Флаг, есть ли что выводить на экран */
4  static unsigned char grafic_out = 1;
5
6 /* Буфер для хранения графической информации */
7  /* Доступ к памяти дисплея намного медленнее */
8  static unsigned char screen[8][128];
9
10/* Очистка буфера экрана */
11void ClearScreen(void){
12      grafic_out = 1;
13      for(unsigned i = 0; i < sizeof(screen); ++i)
14         screen[0][i] = 0x00;
15}
16
17/* Печать нуль терминированной строки в позицию (row, col) */
18void Print(unsigned row, unsigned col, char * byte){
19      grafic_out = 1;
20      if (row > 7) return;
21      while(*byte){
22         for(unsigned i = 0; i < 5 && col < 128; ++i)
23            screen[row][col++] = ansi[*byte][i];
24         if (col < 128) screen[row][col+
25            +] = 0x00;
26         ++byte;
27      }
28}
29
30/* Вывод изображения, если были изменения */
31void ScreenUpdate(void){
32      if          (grafic_out){
33         grafic_out    =    0;
34         LcdScreen(screen);
35      }
36}
```

Задания для самостоятельного выполнения

1. Реализуйте горизонтальную бегущую строку.
2. Реализуйте вертикальную бегущую строку.
3. Реализуйте строку, бегущую по периметру дисплея.
4. Реализуйте счёт в двоичной позиционной системе счисления.
5. Реализуйте счёт в шестнадцатеричной позиционной системе счисления.
6. Добавьте в файл graphics.c функцию вывода переменной типа unsigned int, добавьте объявление в graphics.h.

ЛИТЕРАТУРА

Основные источники:

1. Системное и прикладное программное обеспечение : учебное пособие / составители И. А. Журавлёва, П. К. Корнеев. — Ставрополь : СКФУ, 2017. — 132 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/155253
2. Белугина, С. В. Разработка программных модулей программного обеспечения для компьютерных систем. Прикладное программирование : учебное пособие / С. В. Белугина. — Санкт-Петербург : Лань, 2020. — 312 с. — ISBN 978-5-8114-4496-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/133920
3. Кузнецов, А. С. Системное программирование : учебное пособие / А. С. Кузнецов, И. А. Якимов, П. В. Пересунько. — Красноярск : СФУ, 2018. — 170 с. — ISBN 978-5-7638-3885-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/157574
4. Жулабова, Ф. Т. Системное программирование. Лабораторные работы : учебное пособие / Ф. Т. Жулабова. — Санкт-Петербург : Лань, 2020. — 208 с. — ISBN 978-5-8114-4666-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/140772