Государственное профессиональное образовательное
автономное учреждение Амурской области
«БЛАГОВЕЩЕНСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»
(**ГПОАУ СПО «БПК»**)

Е.П. Мунгалова

Основы программирования на языке Java SE

Учебное пособие

Благовещенск, 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В пособии приведены сгруппированные по занятиям упражнения по программированию на языке Java, а также задачи, контрольные вопросы и теоретический материал, предназначенные для решения студентами, изучающими курс «Программирование на встроенных языках».

Для каждого практического занятия определена область изучаемых Java- технологий и цели практического занятия.

Пособие рекомендуется как для студентов изучающих курс языка Java, так и преподавателей, проводящих по этому курсу практические занятия.

1 ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA

Java является объектно-ориентированным языком программирования. Сравнивая языки Java и C++ можно увидеть существенные различия. Например, программы, написанные на Java, способны работать в разных частях сети и не зависят от платформы, поэтому Java называют интерпретируемым языком. Также в Java нет указателей, которые сложны в использовании и потенциально могут послужить причиной доступа программы к не принадлежащей ей области памяти.

Говоря об объектно-ориентированности языка Java можно выделить основные аспекты:

1. Все является объектом. Программа содержит описания объектов (классов), но это только описания принципа функционирования объектов, описание их основных свойств. Для того, чтобы программа начала работать, необходимо создать в памяти машины экземпляры этих объектов, которые существуют, изменяются во время работы программы и уничтожаются при завершении ее выполнения.

2. Каждый объект имеет свойства - другие объекты или элементарные данные (числа, символы и т.п.). Таким образом, экземпляр объекта фактически является областью памяти, в которой хранятся определенные значения свойств этого объекта.

3. Программа представляет собой совокупность объектов, которые взаимодействуют друг с другом. Каждый объект имеет ряд характерных функций - методов. Доступ к свойствам объекта (изменение, присвоение новых значений и др.) осуществляется путем вызова методов этого объекта. Взаимодействие между объектами реализуется путем вызова их методов друг от друга.

4. Все объекты строго типизированы. Т.е. во время создания экземпляра объекта строго известно, какого типа (класс) этот объект, какими основными свойствами и методами он обладает. (Свойство JAVA).

5. Можно наследовать объекты, которые позволяют описывать новые классы на основе существующих, добавлять к ним новые функции или переопределять существующие.

1.1 Встроенные типы данных

Для написания программ на языке Java необходимо знать его синтаксис.

При объявлении переменной важна последовательность:

тип_данных имя_переменной = начальное_значение_переменной.

Например, int a = 5;. Инициализировать переменные базовых типов можно в программе или при их определении.

В языке программирования Java есть несколько встроенных типов данных:

Тип	**Ключевое слово**	**Размер в байтах**	**Пример**
Байтовый	**byte**	1	123
Короткий целый	**short**	2	12345
Целый	**int**	4	1234567890
Длинный целый	**long**	8	1234567890
Вещественный	**float**	4	123.45f
Вещественный двойной точности	**double**	8	123.456
Символьный	**char**	2	‘a’
Логический	**boolean**	1	true
Строковый	**String**	2	“Hello”

1.2 Операции

Большинство операций в Java просты и интуитивно понятны. К таким операциям относятся: +,-,*,/,<,> и др. Как и в математике операции имеют собственный порядок выполнения и приоритеты. Например, в выражении сначала выполняется умножение, а потом сложение.

1.2.1 Операция присваивания

В отличие от других языков программирования в Java присваивание – это операция, а не оператор. Операция присваивания обозначается символом ‘=’. Она вычисляет значение своего правого операнда и назначает его левому операнду, а также выдает в качестве результата присвоенное значение. Это значение может использоваться другими операциями. Последовательность нескольких назначений выполняется справа налево.

1.2.2 Операция сравнения

К таким операциям относятся: <, >, <=, >=, != и ==. Операция ‘!=’ обозначает не равно, а операция '==' обозначает равенство!

1.2.3 Операции инкремента, декремента

Это операции ++ (инкремент) и -- (декремент) являются сокращенной записью x = x + 1, аналогично и с операцией декремента (--). Данные операции существуют в двух формах – префиксной (++x) и постфиксной (x++). Действие операций одинаково, однако результат разный. Префиксная форма в качестве результата выдает уже измененное на 1 значение операнда, а постфиксная - значение операнда до изменения.

1.2.4 Логические операции

& (логическое умножение). Умножение производится поразрядно, и если у обоих операндов значения разрядов равно 1, то операция возвращает 1, иначе возвращается число 0 (логическая операция «И»).

| **(логическое сложение).** Данная операция также производится по двоичным разрядам, но теперь возвращается единица, если хотя бы у одного числа в данном разряде имеется единица (логическая операция «ИЛИ»).

1.3 Комментарии

Комментарии бывают однострочными и блочными. **Однострочные** комментарии начинаются с ‘//’ и комментируют всего одну строку.
Блочные комментарии начинаются с ‘/*’ и заканчиваются ‘*/’. Все что находится между этими знаками является комментарием.

1.4 Работа с командной строкой

Для получения данных, введенных пользователем, а также для вывода сообщений необходим ряд классов, через которые можно взаимодействовать с консолью. Для взаимодействия с консолью необходим класс **System**. Этот класс располагается в пакете java.lang, который автоматически подключается в программу, поэтому не нужно дополнительно импортировать данный пакет и класс.

1.4.1 Вывод в командную строку

Для создания потока вывода в класс System определен объект **out**. В этом объекте определен метод **println**, который позволяет вывести на консоль некоторое значение с последующим переводом консоли на следующую строку:

```
System.out.println("Hello world");
```

В метод println передается любое значение, как правило, строка, которое надо вывести на консоль. При необходимости можно и не переводить курсор на следующую строку. В этом случае можно использовать метод System.out.**print**(…), который аналогичен println, но исключением является то, что не осуществляет перевод на следующую строку.

```
System.out.print("Hello world");
```

Но с помощью метода System.out.print также можно реализовать перевод каретки на следующую строку. Для этого необходимо использовать escape-последовательность **\n**:

```
System.out.print("Hello world \n");
```

1.4.2 Ввод в командную строку

Для получения консольного ввода в классе System предназначен объект **in**. Однако напрямую через объект System.in не очень удобно работать, поэтому, как правило, используют класс **Scanner**, который, в свою очередь использует System.in.

Класс Scanner имеет ряд методов, которые позволяют получить введенные пользователем значения:

- next(): считывает введенную строку до первого пробела
- nextLine(): считывает всю введенную строку
- nextInt(): считывает введенное число int
- nextDouble(): считывает введенное число double
- hasNext(): проверяет, было ли введено слово
- hasNextInt(): проверяет, было ли введено число int
- hasNextDouble(): проверяет, было ли введено double

Лабораторная работа №1

Ввод/вывод информации

Цель занятия

узнать:

- среду разработки NetBeans;
- основные синтаксические конструкции языка Java;
- способ создания пакетов и программ;
- преобразование типов;
- вывод и ввод результатов в командную строку;
- использование встроенных математических функций.

научиться:

- описывать типы данных правильно;
- конвертировать типы данных;
- вводить данные через командную строку;
- выводить результаты вычислений в командную строку;
- описывать и использовать методы в классах.

Пример:

Создайте программу-анкету. Программа должна задавать вам вопрос, а вы, путем ввода ответа в командную строку, должны на него ответь.

Класс **Input**:

```
package input;
import java.util.Scanner;
public class Input {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print ("Как вас зовут? ");
    //консольный ввод значения типа String:
    String name = s.next();
    System.out.print("Сколько вам лет? ");
    //консольный ввод значения типа int:
    int age = s.nextInt();
     System.out.print("В какой группе вы учитесь?
");
    int group = s.nextInt();
    System.out.print("Есть ли  у вас кот? ");
    String cat = s.next();
  }
```

```
}
```

Задания:

1 задание Написать программу, которая вычисляет ваш вес на Луне

Чем больше планета или звезда, тем сильнее притягивает она другие небесные тела. Масса Луны гораздо меньше массы Земли, и притяжение на Луне составляет всего лишь одну шестую часть земного; это означает, что человек на Луне весит в шесть раз меньше, чем на Земле.

В данной задаче мы напишем программу, которая вычисляет ваш вес на Луне. Исходя из научных данных, сила тяжести на Луне примерно равна 17% от Земной.

Для расчета веса на Луне воспользоваться формулой:

```
ves=(massa*17)/100;
```

2 задание Ввести с консоли несколько целых чисел и определить, какие из них являются четными и нечетными.

3 задание Ввести с консоли несколько целых чисел и определить, какие из них делятся на 3 или на 9.

4 задание Ввести с консоли несколько целых чисел и определить, какие из них делятся на 5 или на 10.

5 задание Ввести с консоли несколько целых чисел и определить, какой из введенных чисел является максимумом, минимумом.

Контрольные вопросы

1. как объявляются переменные?
2. для чего нужен цикл if?
3. какие операторы и операции использовали при выполнении заданий?

2 МЕТОДЫ

Методы в Java - это законченная последовательность действий (инструкций), направленных на решение отдельной задачи. По сути, это те же функции (подпрограммы) более ранних не ООП языков. Класс может содержать несколько методов, все зависит от сложности класса. Название метода всегда завершается двумя круглыми скобками, после которых идет блок кода, обрамлённый фигурными скобками.

Общее определение методов выглядит следующим образом:

```
    [модификаторы]           тип_возвращаемого_значения
название_метода ([параметры]){
        // тело метода
    }
```

Модификаторы и параметры необязательны.

Вызов метода осуществляется в форме:

```
имя_метода(аргументы);
```

После имени метода указываются скобки, в которых перечисляются аргументы - значения для параметров метода.

Например:

```
public class Program{
    public static void main (String args[]){
         hello();
         welcome();
         welcome();
    }
    static void hello(){
        System.out.println("Hello");
    }
    static void welcome(){
        System.out.println("Welcome to Java 10");
    }
}
```

По умолчанию главный класс любой программы на Java содержит метод **main**, который служит точкой входа в программу:

Ключевые слова **public** и **static** являются модификаторами. Далее идет тип возвращаемого значения. Ключевое слово **void** указывает на то, что метод ничего не возвращает. Затем идут название метода - main и в скобках параметры метода - **String[] args**. И в фигурные скобки заключено тело метода - все действия, которые он выполняет.

В примере метод в методе main вызывается один раз метод hello и два раза метод welcome. В этом и заключается одно из преимуществ методов: можно вынести некоторые общие действия в отдельный метод и затем вызывать их многократно в различных местах программы. Поскольку оба метода не имеют никаких параметров, то после их названия при вызове ставятся пустые скобки.

Также следует отметить, что чтобы вызвать в методе main другие методы, которые определены в одном классе с методом main, они должны иметь модификатор **static**.

2.1 Параметры методов

С помощью параметров можно передать в методы различные данные, которые будут использоваться для вычислений. Например:

```
static void sum(int x, int y){
    int z = x + y;
    System.out.println(z);
}
```

Данная функция принимает два параметра - два числа, складывает их и выводит сумму на консоль.

При вызове этого метода в программе необходимо передать на место параметров значения, которые соответствуют типу параметра:

```
public class Program{
    public static void main (String args[]){
        int a = 6;
        int b = 8;
        sum(a, b);  //Сумма равна 14
        sum(3, a);  //Сумма равна 9
        sum(5, 23); //Сумма равна 28
```

```
    }
//Метод вычисляющий сумму:
    static void sum(int x, int y){
        int z = x + y;
        System.out.println(z);
    }
}
```

Поскольку метод sum принимает два значения типа int, то на место параметров надо передать два значения типа int. Это могут быть и числовые литералы, и переменные типов данных, которые представляют тип int или могут быть автоматически преобразованы в тип int. Значения, которые передаются на место параметров, еще называются **аргументами**. Значения передаются параметрам по позиции, то есть первый аргумент первому параметру, второй аргумент - второму параметру и так далее.

2.2 Параметры переменной длины

Метод может принимать параметры переменной длины одного типа. Например, необходимо передать в метод набор чисел и вычислить их сумму, но точно не известно, сколько именно чисел будет передано - 3, 4, 5 или больше. Параметры переменной длины позволяют решить эту задачу:

```
public class Program{
    public static void main (String args[]){
        sum(1, 2, 3);           //Сумма равна 6
        sum(1, 2, 3, 4, 5);     //Сумма равна 15
        sum();                  //Сумма равна 0
    }
    static void sum(int ...nums){
        int result = 0;
        for(int n: nums)
            result += n;
        System.out.println(result);
    }
}
```

Троеточие перед названием параметра int ...nums указывает на то, что он будет необязательным и будет представлять массив. Можно передать в метод sum одно число, несколько чисел, а можно вообще не передавать

никаких параметров. Причем, если нужно передать несколько параметров, то необязательный параметр должен указываться в конце:

```
public static void main(String[] args) {
    sum("Welcome!", 20,10);
    sum("Hello World!");
}
static void sum(String message, int ...nums){
    System.out.println(message);
    int result =0;
    for(int x:nums)
        result+=x;
    System.out.println(result);
}
```

2.3 Оператор return. Результат метода

Методы могут возвращать некоторое значение. Для этого применяется оператор **return**.

```
return возвращаемое_значение;
```

После оператора return указывается возвращаемое значение, которое является результатом метода. Это может быть литеральное значение, значение переменной или какого-то сложного выражения.

Например:

```
public class Program{
    public static void main (String args[]){
        int x = sum(1, 2, 3);
        int y = sum(1, 4, 9);
        System.out.println(x);  //Сумма равна 6
        System.out.println(y);  //Сумма равна 14
    }
    static int sum(int a, int b, int c){
        return a + b + c;
    }
}
```

В методе в качестве типа возвращаемого значения вместо void используется любой другой тип. В данном случае метод sum возвращает значение типа int, поэтому этот тип указывается перед названием метода. Причем если в качестве возвращаемого типа для метода определен любой

другой, отличный от void, то метод обязательно должен использовать оператор return для возвращения значения. При этом возвращаемое значение всегда должно иметь тот же тип, что значится в определении функции. И если функция возвращает значение типа int, то после оператора return стоит целочисленное значение, которое является объектом типа int.

Метод может использовать несколько вызовов оператора return для возвращения разных значений в зависимости от некоторых условий:

```
public class Program{
    public static void main (String args[]){
        System.out.println(daytime(7));
        System.out.println(daytime(13));
        System.out.println(daytime(18));
        System.out.println(daytime(2));
    }
    static String daytime(int hour){
        if (hour >24 || hour < 0)
            return "Invalid data";
        else if(hour > 21 || hour < 6)
            return "Good night";
        else if(hour >= 15)
            return "Good evening";
        else if(hour >= 11)
            return "Good after noon";
        else
            return "Good morning";
    }
}
```

Метод daytime возвращает значение типа String, то есть строку, и в зависимости от значения параметра hour возвращаемая строка будет различаться.

2.4 Класс MATH и математические методы

Класс Math содержит методы для выполнения основных числовых операций, таких как нахождение минимального значения, квадратного корня, среднего значения и т. д.

В классе Math имеются две константы типа double: E и PI. Все методы в классе Math статичны.

Основные математические методы:

Тип	Метод	Описание
int	**max(int a, int b)**	Возвращает больший из аргументов.
int	**min(int a, int b)**	Возвращает меньший из аргументов.
double	**sqrt(double a)**	Возвращает квадратный корень аргумента.
double	**cos(double a)**	Возвращает косинус аргумента.
double	**sin(double a)**	Возвращает синус аргумента.
double	**tan(double a)**	Возвращает тангенс аргумента.
double	**abs (double a)**	Возвращает абсолютное значение (модуль) числа типа double.
double	**exp(double a)**	Возвращает экспоненту аргумента.
double	**log(double a)**	Возвращает натуральный логарифм (по основанию e).
double	**log10(double a)**	Возвращает логарифм по основанию 10.
double	**random()**	Возвращает случайное число от 0.0 (включительно) до 1 (не включительно).

3 УСЛОВНЫЕ КОНСТРУКЦИИ

3.1 Конструкция if/else

Общая форма условного оператора if в Java такая:

```
if (условие) {
//действие(-я), которые выполняются, если условие истинно;
}
else if (условие) {
//действие(-я), которые выполняются, если условие истинно;
} ...
```

Выражение if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет последующее выражение или блок выражений после фигурных скобок.

Например:

```
int number1 = 2;
int number2 = 7;
if(number1< number2){
          System.out.println("Первое число меньше второго");
}
```

Так как, в данном случае первое число меньше второго, то выражение number1< number2 истинно и возвращает значение true. Следовательно, управление переходит в блок кода после фигурных скобок и начинают выполняться содержащиеся там инструкции, а конкретно метод System.out.println("Первое число больше второго");.

Если бы первое число оказалось больше второго или равно ему, то инструкции в блоке if не выполнялись бы. В таком случае применяется блок else. Например:

```
int number1 = 2;
int number2 = 7;
if(number1> number2){
```

```
            System.out.println("Первое число больше
второго");
    }
    else{
            System.out.println("Первое число меньше
второго");
```

3.2 Конструкция switch

Общая форма конструкции switch в Java такая:

```
switch(ВыражениеДляСравнения) {
    case Совпадение1:
    команда;
    break;
case Совпадение2:
    команда;
    break;
case Совпадение3:
    команда;
    break;
default:
    оператор;
    break;
}
```

Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий.

Например:

```
int number = 7;
switch(number){
    case 1:
        System.out.println("число равно 1");
        break;
    case 7:
        System.out.println("число равно 7");
        num++;
        break;
    case 9:
        System.out.println("число равно 9");
        break;
    default:
          System.out.println("число не равно 1, 7,
9");
```

```
}
```

После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case, если совпадение не найдено, то добавляется блок default, но он не обязателен.

В конце блока case ставится оператор break, чтобы избежать выполнения других блоков.

3.3 Циклы

3.3.1 Цикл for

Цикл for имеет следующее формальное определение:

```
for ([инициализация счетчика]; [условие]; [изменение счетчика]){
        // действия
}
```

Рассмотрим стандартный цикл for:

```
for (int i = 1; i < 9; i++){
      System.out.printf("Квадрат числа %d равен %d \n", i, i * i);
}
```

Первая часть объявления цикла - int i = 1 создает и инициализирует счетчик i. Счетчик необязательно должен представлять тип int, он может представлять другой числовой тип, например, float. Перед выполнением цикла значение счетчика будет равно 1. В данном случае это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока i не достигнет 9.

И третья часть - приращение счетчика на единицу, но необязательно увеличивать на единицу, можно уменьшать: i--.

В итоге блок цикла сработает 8 раз, пока значение i не станет равным 9. И каждый раз это значение будет увеличиваться на 1.

3.3.2 Цикл do/while

Конструкция оператора do while:

```
do {
    Тело цикла;
} while (условие выполения);
```

Цикл do сначала выполняет код цикла, а потом проверяет условие в инструкции while. И пока это условие истинно, цикл повторяется. Например:

```
int j = 7;
do{
    System.out.println(j);
    j--;
}
while (j > 0);
```

В данном случае код цикла сработает 7 раз, пока j не окажется равным нулю. Важно отметить, что цикл do гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции while не будет истинно. Так, мы можем написать:

```
int j = -1;
do{
    System.out.println(j);
    j--;
}
while (j > 0);
```

Хотя переменная j изначально меньше 0, цикл все равно один раз выполнится.

3.3.3 Цикл while

Конструкция оператора while:

```
while (Условие выполнения) {
    Тело цикла;
}
```

Цикл while сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

```
int j = 6;
while (j > 0){

    System.out.println(j);
    j--;
}
```

3.4 Операторы break и continue

Оператор break позволяет выйти из цикла в любой его момент, даже если цикл не закончил свою работу.

Например:

```
int number = 7;
while (number < 15){
    if (number == 10) break;
    System.out.println(number);
    number++;
}
```

Для начала идет проверка условия в цикле while, если оно выполняется, то в цикле for сравнивается полученное число с числом 10, если number не равно 10, то оно распечатывается, а затем увеличивается на единицу, а если number равно 10, то оператор break прекращает дальнейшую работу цикла while.

Для того, чтобы цикл не завершался, а просто переходил к следующему элементу, то используется оператор continue.

Например:

```
int[] nums = new int[] { 1, 2, 3, 4, 12, 9 };
for (int i = 0; i < nums.length; i++){

    if (nums[i] > 10)
        continue;
    System.out.println(nums[i]);
}
```

В этом случае, когда выполнение цикла дойдет до числа 12, которое не удовлетворяет условию проверки, то программа просто пропустит это число и перейдет к следующему элементу массива.

Лабораторная работа №2

Расчет и вывод простых уравнений

Цель занятия

узнать:

- способ описания методов;
- как вызывается метод;
- вывод и ввод результатов в командную строку;
- использование встроенных математических функций.

научиться:

- создавать и применять методы;
- вводить данные через командную строку;
- выводить результаты вычислений на командную строку;
- применять встроенные математические функции.

Задания:

1 задание Создать метод вычисляющий факториал любого числа.

Например:

```
public int factorial(int numbers);
```

2 задание Создать методы, вычисляющие уравнения:

1) $ax^2 + bx + c$, на выводе: 5x+7x^2+14;
2) $x + \frac{b}{2a} + \sqrt{x}$, на выводе: 9+(b/2a)+3;
3) $3 - 2\sin sin\, x + 2\left(1 + x + x^2\right)$, на выводе: результат вычислений!
4) $\frac{1+x^3}{1.2-\sqrt{\cos cos\, x}}$, на выводе: результат вычислений!

3 задание Реализовать метод, который ищет корни уравнения вида:

$$ax^2 + bx + c.$$

Контрольные вопросы

1. какую библиотеку использовали для вычисления математических функций?

2. какие методы вы использовали при нахождении значений?
3. Какие бывают методы?

4 КЛАССЫ И ОБЬЕКТЫ

4.1 Классы и объекты

Java - это объектно-ориентированный язык, поэтому такие понятия, как «класс» и «объект», играют ключевую роль в нем. Любая Java-программа может быть представлена как набор объектов, взаимодействующих друг с другом.

Шаблон или описание объекта - это **класс**, а **объект** представляет экземпляр этого класса.

Класс определяется с помощью ключевого слова **class**:

```
class Car{
}
```

В данном примере класс называется Car. После названия класса идут фигурные скобки. Все, что находится между фигурными скобками называется тело класса. В теле класса находятся поля и методы.

Все функциональные возможности класса представлены его членами - полями (поля являются переменными класса), которые хранят состояние объекта и методы, определяющие поведение объекта. Например, класс Car, который представляет машину, может иметь следующее определение:

```
class Car{
    String name;         //наименование
    int power;          //мощность
    void displayInfo(){
              System.out.printf("Name:  %s  \tPower:
%d\n", name, power);
    }
}
```

В классе Car определены два поля: name, который представляет наименование машины, а power - мощность. И также определен метод displayInfo, который ничего не возвращает и просто выводит эти данные на консоль.

Теперь используем данный класс. Для этого определим следующую программу:

```
public class Program{
      public static void main(String[] args) {
         Car BMW;
      }
}
class Car{
    String name;     //наименование
    int power;      //мощность
    void displayInfo(){
              System.out.printf("Name:  %s  \tPower:
%d\n", name, power);
    }
}
```

В методе main определена переменная BMW, которая представляет класс Car. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение null. Пока не возможно использовать данную переменную, поэтому вначале необходимо создать объект класса Car.

4.2 Конструкторы

Кроме обычных методов классы могут определять специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта этого класса. Конструкторы выполняют инициализацию объекта.

Если конструктор не определен в классе, то автоматически создается конструктор без параметров для этого класса. Вышеуказанный класс Car не имеет конструкторов. Поэтому для него автоматически создается конструктор по умолчанию, который можно использовать для создания объекта Car.

Создание объекта будет выглядеть следующим образом:

```
public class Program{
    public static void main(String[] args) {
        Car BMW = new Car(); //создание объекта
        BMW.displayInfo();
        //изменяем наименование и мощность
```

```
            BMW.name = "BMW";
            BMW.power = 140;
            BMW.displayInfo();
        }
    }
    class Car{
        String name;     // наименование
        int power;      // мощность
        void displayInfo(){
                System.out.printf("Name:  %s  \tPower:
%d\n", name, power);
        }
    }
```

Для создания объекта Car используется выражение new Car(). Оператор new выделяет память для объекта Car 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 Car. А переменная BMW получит ссылку на созданный объект.

После создания объекта можно обратиться к переменным объекта Car через переменную BMW и установить или получить их значения, например, BMW.name = "BMW".

В итоге на командной строке получится:

Name: null Power: 0
Name: BMW Power: 140

4.3 Модификаторы доступа

Все поля, методы и свойства имеют модификаторы доступа для того, чтобы задать допустимую область видимости.

В Java используются следующие модификаторы доступа:

- **public:** публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.

- **private:** закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.
- **protected:** такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
- **Модификатор по умолчанию**. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

4.4 Вложенные классы

Язык Java позволяет определять класс в другом классе. Такие классы называются **вложенными массами**. Область вложенного класса ограничена областью действия внешнего класса.

Вложенный класс имеет доступ к членам класса, в который он вложен. Однако внешний класс не имеет доступа к членам вложенного класса. Вложенный класс, который объявлен непосредственно в области видимости его внешнего класса, является его членом.

Существует два типа вложенных классов: статический и нестатический.

Статический вложенный класс - это класс, к которому применяется статический модификатор. Поскольку он статический, он должен ссылаться на нестатические члены своего внешнего класса, используя объект. То есть он не может напрямую ссылаться на нестатические члены своего внешнего класса. Из-за этого ограничения статические вложенные классы используются редко.

```
class Math{
    public static class Factorial{
        private int result;
        private int key;
        public Factorial(int number, int x){
            result=number;
```

```
            key = x;
        }
        public int getResult(){
            return result;
        }
        public int getKey(){
            return key;
        }
    }
    public static Factorial getFactorial(int x){
        int result=1;
        for (int i = 1; i <= x; i++){
            result *= i;
        }
        return new Factorial(result, x);
    }
}
```

Здесь определен вложенный класс для хранения данных о вычислении факториала. Основные действия выполняет метод getFactorial, который возвращает объект вложенного класса. Теперь можно использовать классы в методе main:

```
public static void main(String[] args) {
    Math.Factorial fact = Math.getFactorial(6);
     System.out.printf("Факториал числа %d равен %d
\n", fact.getKey(), fact.getResult());
}
```

4.5 Внутренние классы

Наиболее важным типом вложенного класса является внутренний класс.

Внутренний класс - это нестатический вложенный класс. Он имеет доступ ко всем переменным и методам своего внешнего класса и может напрямую ссылаться на них так же, как и другие нестатические члены внешнего класса. Ссылку на объект внешнего класса из внутреннего класса можно получить с помощью выражения `Внешний_класс.this`.

```
public class Program{
    public static void main(String[] args) {
```

```
            Person tom = new Person("Tom");
            tom.setAccount("qwerty");
        }
    }
    class Person{
        private String name;
        Person(String name){
            this.name = name;
        }
        public void setAccount (String password){
            class Account{
                void display(){
                     System.out.printf("Account Login:
%s \t Password: %s \n", name, password);
                }
            }
            Account account = new Account();
            account.display();
        }
    }
```

Лабораторная работа №3

Создание классов на ЯП Java

Цель занятия

узнать:

- способ создания классов;
- способ создания экземпляра класса;
- создание и применение конструкторов по умолчанию и конструкторы с параметрами;

научиться:

- создавать внутренние и внешние классы;
- обращаться к методам других классов;
- обращаться к другим классам.

Пример

Определить класс Payment (покупка) с внутренним классом, с помощью объектов которого можно сформировать покупку из нескольких товаров.

Класс **Payment:**

```
import java.io.IOException;
import java.util.Scanner;

public class Payment {
    //название покупки private
    String name;
    //перечень товаров
    private Product []prodArray;
    //стоимость покупки
    private int cost;
    //внутренний класс
    private class Product {
        // название товара
        private String productName;
        //стоимость товара
        private int productCost;
        //конструктор по умолчанию
        public Product() {
            super();
            productName = "";
            productCost = 0;
        }
        //конструктор с параметрами
```

```
            public Product(String productName, int
productCost) {
                super();
                this.productName = productName;
                this.productCost = productCost;
            }
            // методы внутреннего класса
            public String getProductName() {
                return this.productName;
            }
            public int getProductCost() {
                return this.productCost;
            }
        }
        //конструктор по умолчанию
        public Payment() {
            super();
            this.name = "";
            this.cost = 0;
        }
        //конструктор с параметрами
        public Payment(String name) {
            super();
            this.name = name;
        }
        public void setPayment()throws IOException {
            this.cost = 0;
            System.out.print("Введите количество
товаров, которое Вы хотите приобрести: ");
            Scanner br = new Scanner(System.in);
            try {
                int dim = br.nextInt();
                prodArray = new Product[dim];
                for(int i = 0; i < dim; i++) {
                    System.out.println("Товар " +
(i+1) +": ");
                    System.out.print("Наименование:
");
                    String str_name = br.next();
                    System.out.print("Цена: ");
                    int prod_cost = br.nextInt();
                    prodArray[i] = new
Product(str_name, prod_cost);
                    this.cost = this.cost +
prodArray[i].productCost;
```

```
                }
            }
            catch(NumberFormatException e) {
                System.out.println("Неверный формат");
            }
            catch(NegativeArraySizeException e) {
                System.out.println("Размерность
массива не может быть < 0");
            }
            catch (NullPointerException e) {
                System.out.println();
                System.out.println("Массив не
создан");
            }
        }
        // печать чека
        public void printCheque()throws IOException {
            try {
                if(this.prodArray.length!= 0) {

System.out.println("============================");
                    System.out.println(" "+
this.name);

System.out.println("============================");
                    for(int i = 0; i <
this.prodArray.length; i++) {
                        System.out.print(i+1);
                        System.out.print("\t" +
this.prodArray[i].productName + "\t");

System.out.print(this.prodArray[i].productCost);
                        System.out.println();
                    }

System.out.println("============================");
                    System.out.print("Общая стоимость:
");
                    System.out.print("\t" +
this.cost);
                    System.out.println();

System.out.println("============================");
                } else {
                    System.out.println();
```

```
                    System.out.println("Массив не
создан");
                }
            }
            catch(NullPointerException e) {
                System.out.println();
                System.out.println("Массив не
создан");
            }
        }
    }
```

Класс **PaymentMain:**

```
    import java.io.IOException;

    public class PaymentMain {
        public static void main(String [] args)throws
IOException {
            try {
                Payment pay1 = new Payment("Первая
покупка");
                pay1.setPayment();
                Payment pay2 = new Payment("Вторая
покупка");
                pay2.setPayment();
                pay1.printCheque();
                pay2.printCheque();
            }
            catch(NumberFormatException e) {
                System.out.println("Неверный формат");
            }
            catch(NullPointerException e) {
                System.out.println("Массив не
создан");
            }
        }
    }
```

Результат выполнения программы:

```
Введите количество товаров, которое Вы хотите приобрести: 3
Товар 1:
Наименование: Milk
Цена: 10000
Товар 2: Наименование: Eggs
Цена: 14000
```

```
Товар 3: Наименование: Sausage
Цена: 87900
Введите количество товаров, которое Вы хотите приобрести: 2
Товар 1:
Наименование: Tea
Цена: 45600
Товар 2: Наименование: Coffee
Кофе Цена: 89000
============================
  Первая покупка
============================
     1     Milk  10000
     2     Eggs  14000
     3  Sausage  87900
============================
Общая стоимость: 111900
============================
============================
  Вторая покупка
============================
     1     Tea  45600
     2  Coffee  89000
============================
Общая стоимость: 134600
============================
```

Задания

!!! Вариант согласно № п.п. (см. журнал) !!!

Вариант	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Задание	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7

1. Создать класс **Account** (счет) с внутренним классом, с помощью объектов которого можно хранить информацию обо всех операциях со счетом (снятие, платежи, поступления).

2. Создать класс **StudentsRecordBook** (зачетная книжка) с внутренним классом, с помощью объектов которого можно хранить информацию о сессиях, зачетах, экзаменах.

3. Создать класс **Department** (отдел фирмы) с внутренним классом, с помощью объектов которого можно хранить информацию обо всех должностях отдела и обо всех сотрудниках, когда-либо занимавших конкретную должность.

4. Создать класс **Catalog** (каталог) с внутренним классом, с помощью объектов которого можно хранить информацию об истории выдач книги читателям.

5. Создать класс **City** (город) с внутренним классом, с помощью объектов которого можно хранить информацию о проспектах, улицах, площадях.

6. Создать класс **CD** (mp3-диск) с внутренним классом, с помощью объектов которого можно хранить информацию о каталогах, подкаталогах и записях.

7. Создать класс **Mobile** (мобильный телефон) с внутренним классом, с помощью объектов которого можно хранить информацию о моделях телефонов и их свойствах.

8. Создать класс **Shop** (магазин) с внутренним классом, с помощью объектов которого можно хранить информацию об отделах, товарах и услугах.

9. Создать класс **Computer** (компьютер) с внутренним классом, с помощью объектов которого можно хранить информацию об операционной системе, процессоре и оперативной памяти.

10. Создать класс **TVProgram** (программа передач) с внутренним классом, с помощью объектов которого можно хранить информацию о названии телеканалов и программ.

Контрольные вопросы

1. какие классы называются внутренними, внешними?
2. являете ли конструктор методом?
3. какие модификаторы доступа вы использовали при обращении к методам другого класса?

Лабораторная работа №4

Логические и условные операторы на ЯП Java (IF)

Цель занятия

узнать:

- как работает условный оператор;
- какая конструкция у условного оператора if/else;
- как зависит поиск решения от поставленного условия.

научиться:

- задавать условия;
- поиску решений различных задач;
- определять различные действия от заданных условий.

Задания:

1 задание Необходимо написать программу, где бы пользователю предлагалось ввести число на выбор: 5, 7 или 15, а программа должна сказать, какое число ввёл пользователь: 5, 7, или 15.

2 задание Написать программу, которая загадает число, а пользователь должен угадать загаданное число. Например, в программе загадано число 32, пользователь вводит в консоль любое число и если оно совпадает с загаданным программой числом, то выводится сообщение «Ты угадал!», а если загаданное число не совпало, то появится сообщение «Неправильно».

3 задание Даны 4 числа. Все отрицательные среди них числа заменить на 0.

4 задание Дано 6 чисел. Найти разность между наибольшим и наименьшим .

5 задание Дано 6 случайных чисел. Упорядочить по возрастанию, по убыванию.

Контрольные вопросы

1. какие значения должно принимать условие?
2. обязательна ли конструкция else?
3. сколько условных операторов может быть?

4. всегда ли ставятся скобки и какие в условном операторе?

Лабораторная работа №5

Логические и условные операторы на ЯП Java (CASE)

Цель занятия

узнать:

- как работает условный оператор case;
- какая конструкция у условного оператора case;
- как зависит поиск решения от поставленного условия.

научиться:

- задавать условия;
- определять решение различных задач;
- определять различные действия от заданных условий.

1 задание Вводится целое число С. Если -3<=С<=3 вывести величину числа в словесной форме с учетом знака, в противном случае - предупреждающее сообщение и повторный ввод.

2 задание Ввести номер месяца. Программа должна определить число какого месяца ввел пользователь. Например, ввели число 5, программа распечатывает сообщение «Вы выбрали Май месяц».

3 задание Ввести значение, представляющее день недели. Программа должна определить название этого дня.

4 задание Загадайте загадку пользователю, например, «Сто одежек и все без застежек». Программа будет считывать с консоли ответ, если он верен, то выводится сообщение «Правильно!», а если не верно, то предлагается повторный ввод, если пользователь ввел текст «Сдаюсь», то программа выводит ответ на загадку.

Контрольные вопросы

1. зачем применяется условный оператор CASE?
2. задается ли значение по умолчанию?
3. сколько условных операторов CASE может быть?
4. зачем нужен оператор BREAK?

5 задание

Лабораторная работа №6

Логические и условные операторы на ЯП Java (FOR)

Цель занятия

узнать:

- способ описания методов;
- как вызывается метод;
- вывод и ввод результатов в командную строку;
- использование встроенных математических функций.

научиться:

- создавать и применять методы;
- вводить данные через командную строку;
- выводить результаты вычислений на командную строку;
- применять встроенные математические функции.

1 задание Необходимо вывести на экран числа от 1 до 10.

2 задание Необходимо вывести на экран числа от 15 до 1.

3 задание Создать программу, которая выводит на экран таблицу умножения на 6, на 9.

4 задание Создать программу, которая будет проверять попало ли случайно выбранное из отрезка [5;155] целое число в интервал (25;100) и сообщать результат на экран.

5 задание Вычислить сумму элементов фрагмента последовательности 2, 4, 6, 8, …, 98, 100.

Контрольные вопросы

1. можно ли применять операции инкремента и декремента?
2. виды цикла FOR?
3. обязательны ли все 3 конструкции в круглых скобках после FOR?

Лабораторная работа №7

Ораторы цикла в ЯП Java. Оператор BREAK и CONTINUE

Цель занятия

узнать:

- применение операторов break и continue;
- в каких случаях лучше применять оператор break, а в каких continue;
- как работают операторы.

научиться:

- работать с операторами break и continue;
- понимать их предназначение.

1 задание Дано 10 чисел. Написать программу, которая будет последовательно распечатывать числа от 1, пока не достигнет числа 6.

2 задание Дано 100 чисел. Написать программу, которая будет последовательно распечатывать числа от 1, пока не достигнет числа 23.

3 задание Написать программу, которая распечатывает в каждой строке два числа.

4 задание Написать программу, которая распечатывает в каждой строке три числа.

Контрольные вопросы

1. как работает оператор BREAK?
2. как работает оператор CONTINUE?
3. зачем нужны эти операторы?
4. что будет если в бесконечном цикле не применить оператор BREAK?

5 МАССИВЫ

5.1 Одномерные массивы

Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:

```
тип_данных название_массива[];
//либо
тип_данных[] название_массива;
```

Например, определены массивы чисел:

```
int nums[];
int[] nums2;
```

После объявления массива можно инициализовать его:

```
int nums[];
nums = new int[4];  // массив из 4 чисел
```

Создание массива производится с помощью следующей конструкции: `new тип_данных[количество_элементов]`, где **new** - ключевое слово, выделяющее память для указанного в скобках количества элементов.

Например, nums = new int[4]; - в этом выражении создается массив из четырех элементов int, и каждый элемент будет иметь значение по умолчанию - число 0.

Также можно сразу при объявлении массива инициализировать его:

```
int nums[] = new int[4];     // массив из 4 чисел
int[] nums2 = new int[5];    // массив из 5 чисел
```

При подобной инициализации все элементы массива имеют значение по умолчанию. Для числовых типов (в том числе для типа char) это число 0, для типа boolean это значение false, а для остальных объектов это значение null. Например, для типа int значением по умолчанию является число 0, поэтому выше определенный массив nums будет состоять из четырех нулей.

Однако также можно задать конкретные значения для элементов массива при его создании:

```
// эти два способа равноценны
int[] nums = new int[] { 1, 2, 3, 5 };
int[] nums2 = { 1, 2, 3, 5 };
```

Стоит отметить, что в этом случае в квадратных скобках не указывается размер массива, так как он вычисляется по количеству элементов в фигурных скобках.

После создания массива можно обратиться к любому его элементу по индексу, который передается в квадратных скобках после названия переменной массива:

```
int[] nums = new int[4];
// устанавливаются значения элементов массива
nums[0] = 1;
nums[1] = 2;
nums[2] = 4;
nums[3] = 100;
// вывод значения третьего элемента массива
System.out.println(nums[2]);    //выведет 4
```

Индексация элементов массива начинается с 0, поэтому в данном случае, чтобы обратиться к четвертому элементу в массиве, нужно использовать выражение nums[3].

И так как массив определен только для 4 элементов, то не возможно обратиться, например, к шестому элементу: nums[5] = 5;. Если так попытаться сделать, то Java выдаст ошибку.

5.2 Длина массива

Важнейшее свойство, которым обладают массивы, является свойство **length**, возвращающее длину массива, то есть количество его элементов:

```
int[] nums = {1, 2, 3, 4, 5};
int length = nums.length;   // 5
```

Нередко бывает неизвестным последний индекс, и чтобы получить последний элемент массива, можно воспользоваться этим свойством:

```
int last = nums[nums.length-1];
```

5.3 Многомерные массивы

Кроме одномерных массивов также бывают и многомерные массивы. Наиболее известный многомерный массив - таблица, представляющая двухмерный массив:

```
int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };
int[][] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Визуально оба массива можно представить следующим образом:

Одномерный массив nums1

0	1	2	3	4	5

Двухмерный массив nums2

0	1	2
3	4	5

Поскольку массив nums2 двухмерный, он представляет собой простую таблицу. Его также можно было создать следующим образом: int[][] nums2 = new int[2][3];. Количество квадратных скобок указывает на размерность массива. В первых квадратных скобках укахывается количество строк, а во вторых – количество столбцов. И также, используя индексы, можно использовать элементы массива в программе:

```
// устанавливается элемент первого столбца второй строки
nums2[1][0]=44;
System.out.println(nums2[1][0]);
```

Объявление трехмерного массива могло бы выглядеть так:

```
int[][][] nums3 = new int[2][3][4];
```

5.4 foreach

Специальная версия цикла for предназначена для перебора элементов в наборах элементов, например, в массивах и коллекциях. Она аналогична действию цикла foreach, который имеется в других языках программирования.

Формальное объявление:

```
for (тип_данных название_переменной : контейнер){
    // действия
}
```

Например:

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
for (int i : array){
    System.out.println(i);
}
```

В качестве контейнера в данном случае выступает массив данных типа int. Затем объявляется переменная с типом int.

То же самое можно было бы сделать и с помощью обычной версии for:

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i < array.length; i++){
    System.out.println(array[i]);
}
```

В то же время эта версия цикла for более гибкая по сравнению с `for(int i:array)`. В частности, в этой версии можно изменять элементы:

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
for (int i=0; i<array.length;i++){
    array[i] = array[i] * 2;
    System.out.println(array[i]);
}
```

Перебор многомерных массивов в цикле:

```
int[][] nums = new int[][] {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
for (int i = 0; i < nums.length; i++){
    for(int j=0; j < nums[i].length; j++){
       System.out.printf("%d ", nums[i][j]);
    }
    System.out.println();
}
```

Сначала создается цикл для перебора по строкам, а затем внутри первого цикла создается внутренний цикл для перебора по столбцам конкретной строки.

5.5 Сортировка массивов

Одной из частых задач при работе с массивами является сортировка массива.

Сортировкой массива называется процесс упорядочивания элементов массива по возрастанию или по убыванию.

5.5.1 Сортировка выбором

Это один из самых простых алгоритмов сортировки массива. Его простота реализации сказывается на скорости работы такого алгоритма.

Шаги алгоритма:

1. Находится номер минимального значения в текущем списке;
2. Производится обмен этого значения со значением первой не отсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
3. Теперь сортируется хвост списка, исключив из рассмотрения уже отсортированные элементы.

```
public static void selectionSort(int[] arr){
    for (int i = 0; i < arr.length; i++) {
        /*Предполагается, что первый элемент (в каждом
                    подмножестве элементов) является
минимальным
        */
        int min = arr[i];
        int min_i = i;
             /*В оставшейся части подмножества ищется
элемент, который меньше предположенного минимума*/
        for (int j = i+1; j < arr.length; j++) {
            //Если найден, запоминается его индекс
            if (arr[j] < min) {
                min = arr[j];
                min_i = j;
            }
        }
```

```
            /*Если нашелся элемент, меньший, чем на
текущей позиции, то они меняются местами*/
        if (i != min_i) {
            int tmp = arr[i];
            arr[i] = arr[min_i];
            arr[min_i] = tmp;
        }
     }
}
```

5.5.2 Сортировка пузырьком

Алгоритм проходит массив от начала и до конца, сравнивая попарно соседние элементы. Если элементы стоят в неправильном порядке, то они меняются местами, таким образом, после первого прохода на конце массива оказывается максимальный элемент (для сортировки по возрастанию). Затем проход массива повторяется, и на предпоследнем месте оказывается другой наибольший после максимального элемент и т.д. В итоге, наименьший элемент постепенно перемещается к началу массива («всплывает» до нужной позиции как пузырёк в воде).

```
    public static void bubbleSort(int[] arr){
         /*Внешний цикл каждый раз сокращает фрагмент
массива, так как внутренний цикл каждый раз ставит в
конец фрагмента максимальный элемент*/
        for(int i = arr.length-1 ; i > 0 ; i--){
            for(int j = 0 ; j < i ; j++){
                /*Сравниваются попарно элементы,
                  если они имеют неправильный порядок,
                  то они меняются местами*/
                if( arr[j] > arr[j+1] ){
                    int tmp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = tmp;
                }
            }
        }
    }
```

5.5.3 Сортировка вставками

На каждом шаге алгоритма выбирается один из элементов входных данных и вставляется на нужную позицию в уже отсортированном списке до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора. Обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

```
public static void insertSort(int[] arr){
    int key;
    for (int i = 1; i < arr.length; i++) {
        key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] < key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Время сортировки вставками зависит от массива. Чем больше сортируемый массив, тем больше может понадобиться времени для его сортировки.

Лабораторная работа №8

Массивы в ЯП Java. Сортировка массивов в ЯП Java. Многомерные массивы в ЯП Java

Цель занятия

узнать:

- как задать размерность массиву;
- способы создания массивов;
- о различных видах сортировки;
- разницу между одномерным и многомерным массивом.

научиться:

- создавать и применять массивы;
- сортировать массивы различными способами;
- обрабатывать одномерный и многомерный массивы.

1 задание Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести четные и нечетные числа.

2 задание Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести максимум и минимум.

3 задание Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести среднее арифметическое введенных чисел.

Задания по вариантам

!!! Вариант согласно № п.п. (см. журнал) !!!

Вариант	1	2	3	4	5	6	7	8	9	10
Задание	1,6	2,4	3,5	1,5	2,6	3,4	1,6	2,4	3,5	1,5

1. Создать массив целых чисел и сделать сортировку выбором.
2. Создать массив целых чисел и сделать сортировку пузырьком.
3. Создать массив целых чисел и сделать сортировку вставками.
4. Создать двумерный массив целых чисел и сделать сортировку выбором.
5. Создать двумерный массив целых чисел и сделать сортировку пузырьком.
6. Создать двумерный массив целых чисел и сделать сортировку вставками.

Контрольные вопросы

1. какие бывают массивы?
2. как объявляются массивы?
3. всегда ли нужно указывать размерность?
4. как можно задать начальные значения массивам?
5. зачем нужен цикл FOR при работе с массивами?
6. хватит ли одного цикла FOR для прохождения двумерного массива?

6 КОЛЛЕКЦИИ

Для хранения наборов данных в Java предназначены массивы. Однако их не всегда удобно использовать, прежде всего потому, что они имеют фиксированную длину. Эту проблему в Java решают коллекции. Однако суть не только в гибких по размеру наборах объектов, но в и том, что классы коллекций реализуют различные алгоритмы и структуры данных, например, такие как стек, очередь, дерево и ряд других.

Коллекция— это группа элементов с операциями добавления, извлечения и поиска элемента. Механизм работы операций существенно различается в зависимости от типа коллекции.

Классы коллекций располагаются в пакете java.util, поэтому перед применением коллекций следует подключить данный пакет.

В Java существует широкая палитра классов коллекций - списки, множества, очереди, отображения и другие, среди которых можно выделить следующие:

- ArrayList: простой список объектов;
- LinkedList: представляет связанный список;
- TreeSet: набор отсортированных объектов в виде дерева;
- PriorityQueue: очередь приоритетов;
- HashMap: структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение;
- TreeMap: структура данных в виде дерева, где каждый элемент имеет уникальный ключ и некоторое значение.

6.1 Интерфейс Collection

Интерфейс Collection является базовым для всех коллекций, определяя основной функционал:

```
    public interface Collection<E> extends
Iterable<E>{
          // определения методов
    }
```

Интерфейс Collection является обобщенным и расширяет интерфейс Iterable, поэтому все объекты коллекций можно перебирать в цикле по типу for-each.

Среди методов интерфейса Collection можно выделить следующие:

- boolean **add** (E item): добавляет в коллекцию объект item. При удачном добавлении возвращает true, при неудачном – false;
- boolean **addAll** (Collection<? extends E> col): добавляет в коллекцию все элементы из коллекции col. При удачном добавлении возвращает true, при неудачном – false;
- void **clear ()**: удаляет все элементы из коллекции;
- boolean **contains** (Object item): возвращает true, если объект item содержится в коллекции, иначе возвращает false;
- boolean **isEmpty ()**: возвращает true, если коллекция пуста, иначе возвращает false;
- boolean **remove** (Object item): возвращает true, если объект item удачно удален из коллекции, иначе возвращается false;
- boolean **removeAll** (Collection<?> col): удаляет все объекты коллекции col из текущей коллекции. Если текущая коллекция изменилась, возвращает true, иначе возвращается false;
- boolean **retainAll** (Collection<?> col): удаляет все объекты из текущей коллекции, кроме тех, которые содержатся в коллекции col. Если текущая коллекция после удаления изменилась, возвращает true, иначе возвращается false;
- int **size ()**: возвращает число элементов в коллекции;
- Object[] **toArray** (): возвращает массив, содержащий все элементы коллекции;

Все эти и остальные методы, которые имеются в интерфейсе Collection, реализуются всеми коллекциями, поэтому в целом общие принципы работы с коллекциями будут одни и те же. Единообразный интерфейс упрощает понимание и работу с различными типами коллекций. Так, добавление элемента будет производиться с помощью метода add, который принимает добавляемый элемент в качестве параметра. Для удаления вызывается метод remove(). Метод clear будет очищать коллекцию, а метод size возвращать количество элементов в коллекции.

6.2 Класс ArrayList и интерфейс List

Для создания простых списков применяется интерфейс List, который расширяет функциональность интерфейса Collection.

Некоторые наиболее часто используемые методы интерфейса List схожи с методами интерфейса Cjllection:

- void **add**(int index, E obj): добавляет в список по индексу index объект obj;
- boolean **addAll**(int index, Collection<? extends E> col): добавляет в список по индексу index все элементы коллекции col. Если в результате добавления список был изменен, то возвращается true, иначе возвращается false;
- E **get**(int index): возвращает объект из списка по индексу index;
- int **indexOf**(Object obj): возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1;
- int **lastIndexOf**(Object obj): возвращает индекс последнего вхождения объекта obj в список. Если объект не найден, то возвращается -1;
- E **remove**(int index): удаляет объект из списка по индексу index, возвращая при этом удаленный объект;
- E **set**(int index, E obj): присваивает значение объекта obj элементу, который находится по индексу index;

- void **sort**(Comparator<? super E> comp): сортирует список с помощью компаратора comp;
- и др.

По умолчанию в Java есть встроенная реализация этого интерфейса - класс ArrayList.

Класс ArrayList представляет обобщенную коллекцию, которая наследует свою функциональность от класса AbstractList и применяет интерфейс List. Проще говоря, ArrayList представляет простой список, аналогичный массиву, за тем исключением, что количество элементов в нем не фиксировано.

ArrayList имеет следующие конструкторы:

- ArrayList(): создает пустой список;
- ArrayList(Collection <? extends E> col): создает список, в который добавляются все элементы коллекции col.;
- ArrayList (int capacity): создает список, который имеет начальную емкость capacity.

Емкость в ArrayList представляет размер массива, который будет использоваться для хранения объектов. При добавлении элементов фактически происходит перераспределение памяти - создание нового массива и копирование в него элементов из старого массива. Изначальное задание емкости ArrayList позволяет снизить подобные перераспределения памяти, тем самым повышая производительность.

Пример использования класса ArrayList:

```
import java.util.ArrayList;
public class Program{
      public static void main(String[] args) {
        ArrayList<String> people = new
ArayList<String>();
          // добавление в список ряд элементов
          people.add("Tom");
          people.add("Alice");
          people.add("Kate");
          people.add("Sam");
```

```
            // добавляется элемент по индексу 1
            people.add(1, "Bob");
            // получение 2-го объекта
            System.out.println(people.get(1));
          // установка нового значения для 2-го
объекта
            people.set(1, "Robert");
            System.out.printf("ArrayList has %d
elements \n", people.size());
            for(String person : people){
                System.out.println(person);
            }
            // проверка на наличие элемента
            if(people.contains("Tom")){
                System.out.println("ArrayList contains
Tom");
            }
            // удаление конкретного элемента
            people.remove("Robert");
            // удаление по индексу
            people.remove(0);
            Object[] peopleArray = people.toArray();
            for(Object person : peopleArray){
                System.out.println(person);
            }
        }
    }
```

Консольный вывод программы:

```
Bob
ArrayList has 5 elements
Tom
Robert
Alice
Kate
Sam
ArrayList contains Tom
Alice
Kate
Sam
```

Здесь объект ArrayList типизируется классом String, поэтому список будет хранить только строки. Для добавления вызывается метод add. С его помощью можно добавлять объект в конец списка: people.add("Tom"). Также можно добавить объект на определенное место в списке, например, добавить объект на второе место (то есть по индексу 1, так как нумерация начинается с нуля): people.add(1, "Bob")

Метод size() позволяет узнать количество объектов в коллекции.

Проверку на наличие элемента в коллекции производится с помощью метода contains.

Удаление элемента из коллекции происходит с помощью метода remove. И так же, как и с добавлением, можно удалить либо конкретный элемент people.remove("Tom");, либо элемент по индексу people.remove(0); - удаление первого элемента.

Получить определенный элемент по индексу возможно с помощью метода get(): String person = people.get(1);, а установить элемент по индексу с помощью метода set: people.set(1, "Robert");

С помощью метода toArray() можно преобразовать список в массив объектов.

Лабораторная работа №9

Коллекции ЯП Java

Цель занятия

узнать:

- способы использования коллекций при разработке java-приложений;
- способы создания и работы с коллекциями;
- как можно применять коллекции вместо массивов.

научиться:

- способам использования коллекций при разработке java-приложений;
- обрабатывать различный поток данных;
- работать с коллекциями как с массивами.

Пример

Вычислить сколько раз каждая буква встречается в тексте.

```
import java.util.HashMap;
import java.util.*;
public class Main {
   public static void main(String[] args) {
      String txt = "лабораторная работа";
            HashMap<Character, Integer> map = new
HashMap<Character, Integer>(40);
      for(int i = 0; i < txt.length(); ++i) {
          char c = txt.charAt(i);
          //проверяем является ли символ буквой
          if(Character.isLetter(c)) {
             if(map.containsKey(c)) {
                 map.put(c, map.get(c) + 1);
             } else { map.put(c, 1); }
          }
      }
      //вывод на экран букв с частотой их появления
             for(Entry<Character, Integer> entry :
map.entrySet()) {
                                System.out.println("буква:
"+entry.getKey()+" кол - во: "+entry.getValue());
      }
   }
}
```

Задания

1 задание Сгенерируйте массив целых чисел, используя класс Random(). Используя методы классов-коллекций:

1) отсортируйте массив;
2) найдите максимум, минимум и сумму чисел массива;
3) создайте массив, содержащий все положительные числа массива;
4) удалите из массива все нечетные числа.

2 задание Ввести строки, записать их в стек. Вывести строки в обратном порядке.

3 задание Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные - в начало этого списка.

Контрольные вопросы

1. чем коллекции отличаются от массивов?
2. нужно ли указывать размерность?
3. можно ли задать начальные значения коллекциям?
4. при работе с большим потоком данных удобнее работать с коллекциями или с массивами?

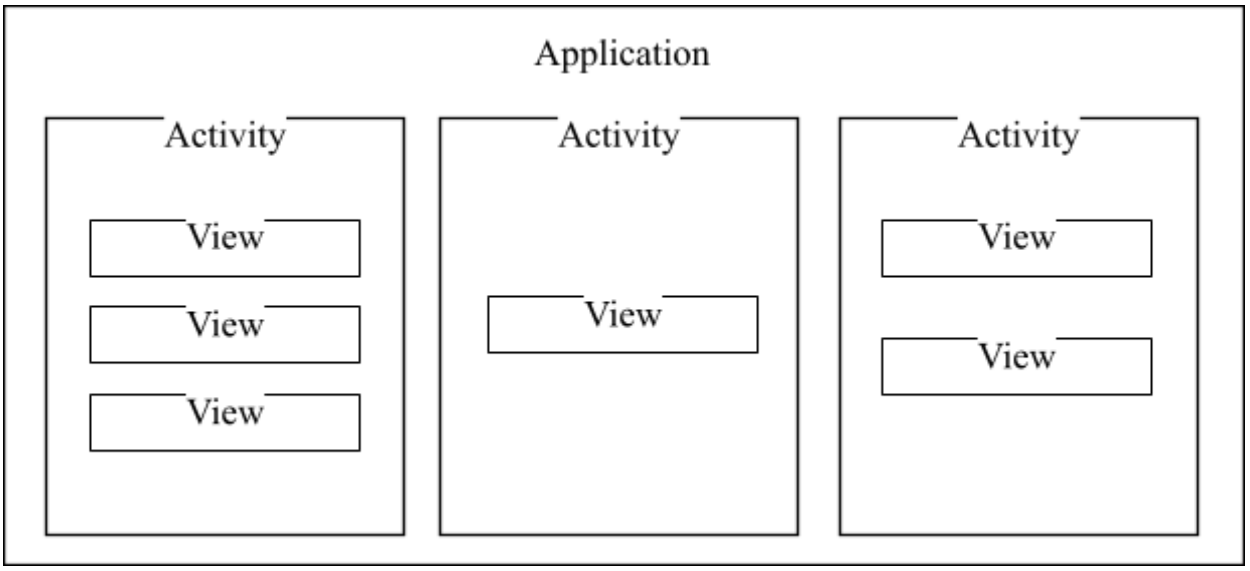
7 ANDROID

7.1 Элементы экрана и их свойства

Если проводить аналогию с Windows, то приложение состоит из окон, называемых **Activity**. В конкретный момент времени обычно отображается одно Activity и занимает весь экран, а приложение переключается между ними. В качестве примера можно рассмотреть почтовое приложение. В нем одно Activity – список писем, другое – просмотр письма, третье – настройки ящика. При работе вы перемещаетесь по ним.

Содержимое Activity формируется из различных компонентов, называемых **View**. Самые распространенные View - это кнопка, поле ввода, чекбокс и т.д.

Примерно это можно изобразить так:



Необходимо заметить, что View обычно размещаются в ViewGroup. Самый распространенный пример ViewGroup - это Layout. Layout бывает различных типов и отвечает за то, как будут расположены его дочерние View на экране (таблицей, строкой, столбцом и т.д.).

7.2 Графический интерфейс приложения

При создании нового проекта автоматически создается по умолчанию файл activity_main.xml, который содержит определение графического интерфейса приложения.

Если файл открыт в режиме дизайнера, а справа в Android Studio отображается дизайн приложения, то следует переключить вид файла в текстовый (рисунок 2). Для переключения режима - из текстового в графический и обратно внизу есть две кнопки Design и Text.

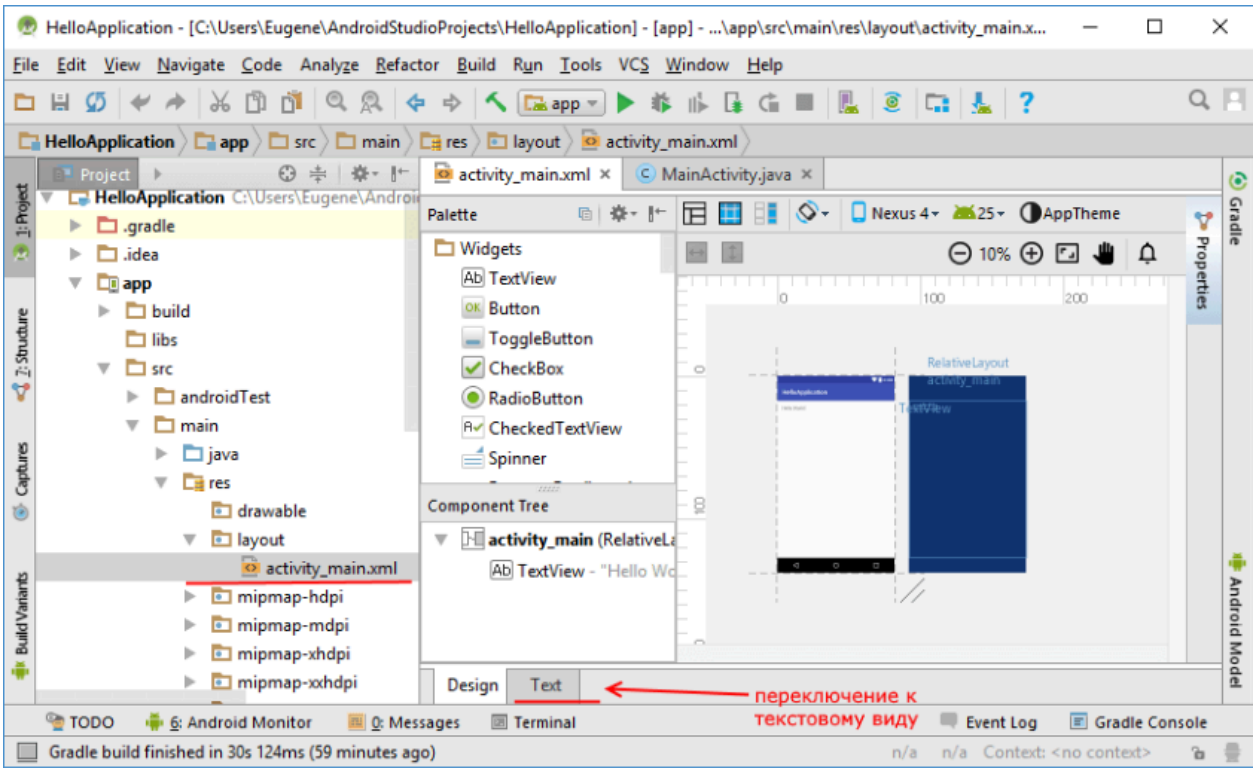


Рисунок 2 - Файл activity_main.xml в режиме Дизайнера

7.3 Основные элементы управления

7.3.1 TextView

Для простого вывода текста на экран предназначен элемент **TextView**. Он просто отображает текст без возможности его редактирования. Некоторые его основные атрибуты:

- android:text: устанавливает текст элемента;

- android:textSize: устанавливает высоту текста, в качестве единиц измерения для указания высоты используются sp;
- android:background: задает фоновый цвет элемента в виде цвета в шестнадцатиричной записи или в виде цветового ресурса;
- android:textColor: задает цвет текста;
- android:textAllCaps: при значении true делает все символы в тексте заглавными;
- android:textDirection: устанавливает направление текста. По умолчанию используется направление слева направо, но с помощью значения rtl можно установить направление справа налево;
- android:textAlignment: задает выравнивание текста. Может принимать следующие значения;
- android:fontFamily: устанавливает тип шрифта. Может принимать следующие значения.

7.3.2 EditText

Элемент **EditText** является подклассом класса TextView. Он также представляет текстовое поле, но теперь уже с возможностью ввода и редактирования текста. Таким образом, в EditText можно использовать все те же возможности, что и в TextView.

Так как элемент EditText использует те же атрибуты, что и TextView, то следует отметить характерные только для EditText атрибуты:

- android:hint. Он позволяет задать текст, который будет отображаться в качестве подсказки, если элемент EditText пуст;
- android:inputType, который позволяет задать клавиатуру для ввода.

7.3.3 Button

Одним из часто используемых элементов являются кнопки, которые представлены классом android.widget.Button.

Ключевой особенностью кнопок является возможность взаимодействия с пользователем через нажатия.

Некоторые ключевые атрибуты, которые можно задать у кнопок:

- text: задает текст на кнопке;
- textColor: задает цвет текста на кнопке;
- background: задает фоновый цвет кнопки;
- textAllCaps: при значении true устанавливает текст в верхнем регистре. По умолчанию как раз и применяется значение true;
- onClick: задает обработчик нажатия кнопки.

Лабораторная работа №10

Установка Android Studio. Разработка программ для ОС Android

Цель занятия

узнать:

- о способе разработки Android-приложений;
- о среде разработки Android Studio;
- про установку разработанного приложения на телефон.

научиться:

- разрабатывать Android-приложения;
- устанавливать Android-приложения на телефон;

создавать Android-приложения в Android Studio.

В первую очередь для создания приложений нужно загрузить и установить пакет JDK 8, который необходим для разработки на языке Java. JDK 8 можно найти на сайте компании Oracle. Ссылка на пакет JDK 8: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Существуют разные среды разработки для Android, но рекомендуемой средой разработки является Android Studio. Загрузить файл установщика можно с официального сайта: http://developer.android.com/sdk/index.html.

Для скачивания пакета установки для OS Windows надо нажать на кнопку "Download Android Studio":

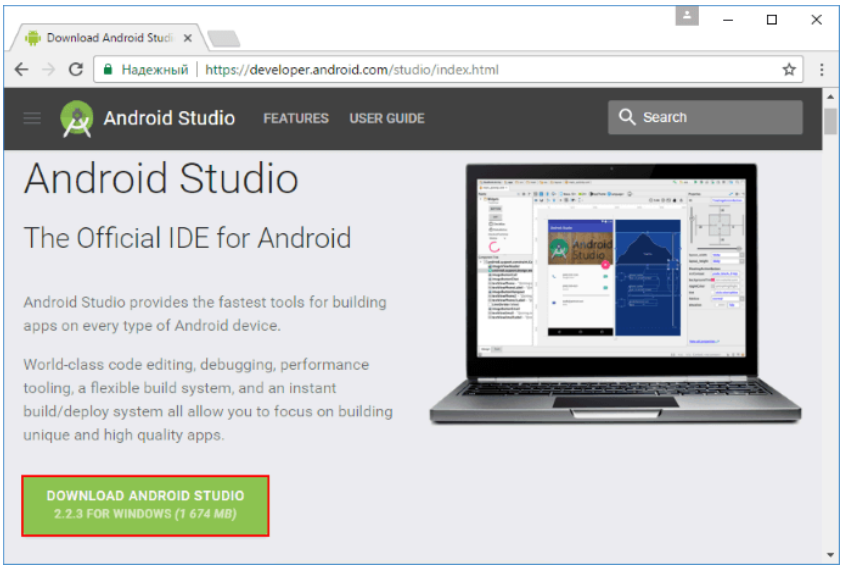


Рисунок 3 - Официальный сайт для скачивания Android Studio

Установка Android Studio. Ничего необычного в установке нет - обычный диалог инсталлятора. В процессе нужно будет ответить лишь на один важный вопрос, и то это опционально:

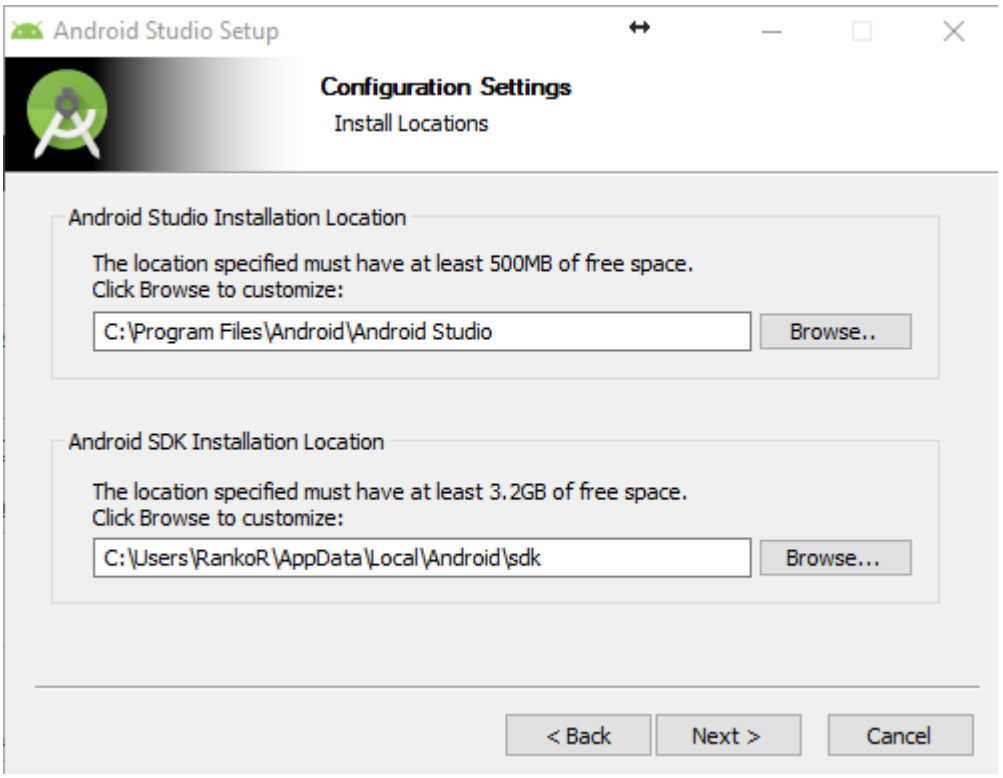


Рисунок 4 - Этап выбора пути установки

Как следует из скриншота (рисунок 4), установщик спрашивает, куда ставить студию, и куда ставить SDK. С SDK нужно быть внимательным.
Во-первых, для установки SDK нужно минимум 3.2 GB места на диске.
Во-вторых, путь установки SDK и Android Studio должен содержать английский алфавит.

После этого понадобится стандартно несколько раз нажать на кнопочку «далее», и на этом установка Android Studio завершена.

Запуск Android Studio. При первом запуске вам будет предложено установить в диалоговых окнах несколько параметров конфигурации. В первом диалоговом окне основное внимание уделяется импорту настроек из ранее установленной версии Android Studio:

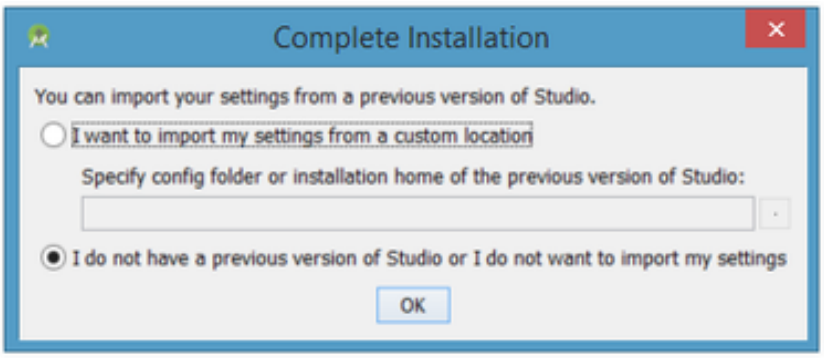


Рисунок 5 - Параметры импорта

Можно принять настройки по умолчанию и нажать на кнопку «ОК». После этого Android Studio выведет диалоговое окно «Мастера установки»:

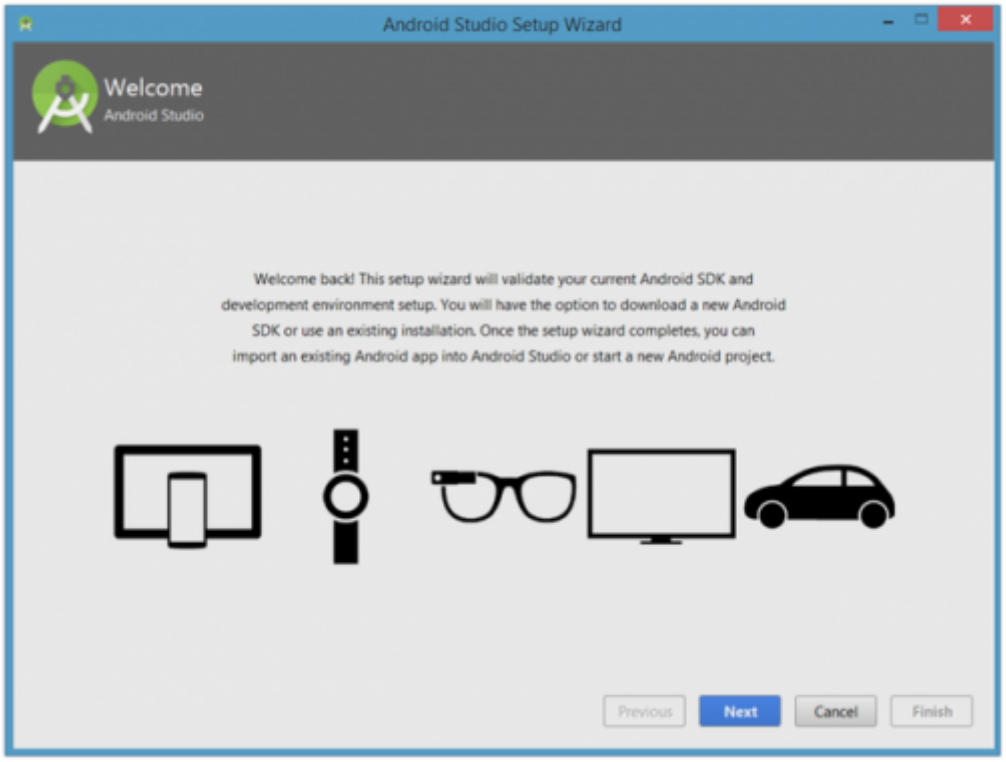


Рисунок 6 - Проверка настроек Android SDK и среды разработки

После нажатия кнопки «Далее», «Мастер установки» предложит выбрать тип установки компонентов SDK. На данный момент я рекомендую использовать стандартную конфигурацию:

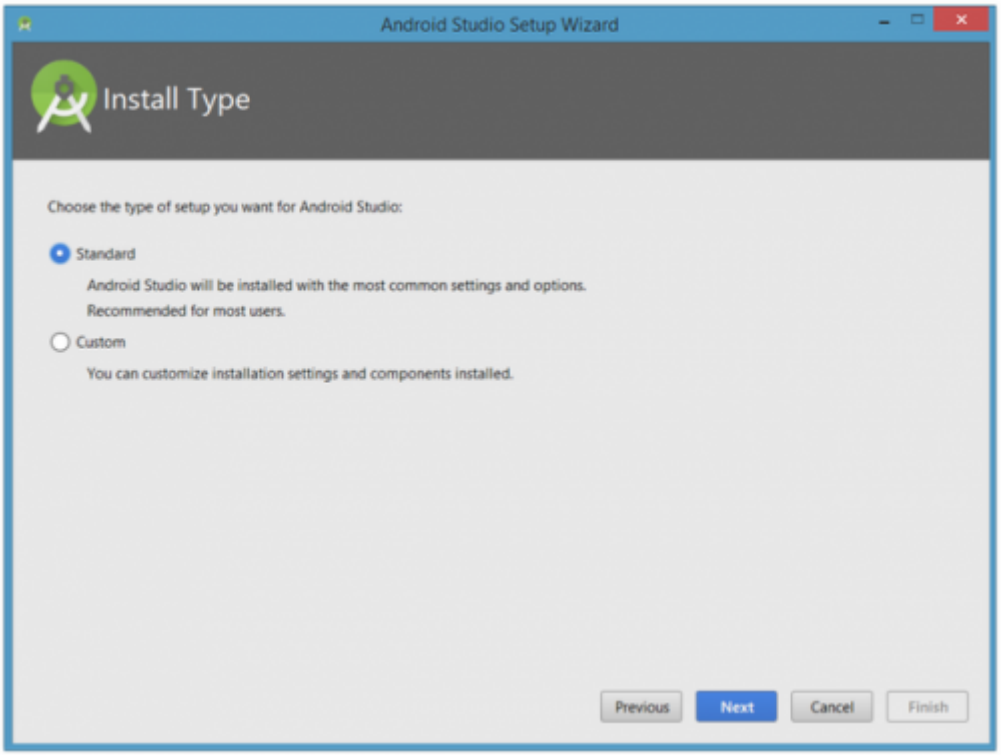


Рисунок 7 Выбор типа установки

Нажмите кнопку «Далее» и подтвердите выбранные настройки. Затем нажмите кнопку «Готово», чтобы продолжить:

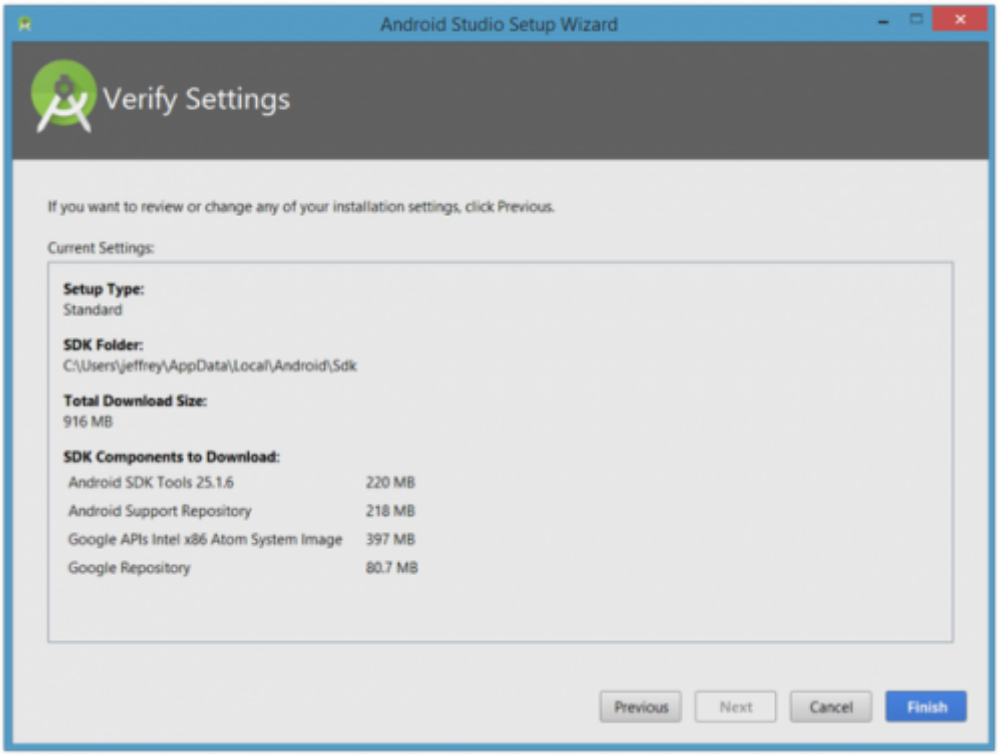


Рисунок 8 - Просмотр настроек

Нажмите кнопку «Готово», чтобы завершить работу «Мастера установки». После этого вы увидите диалоговое окно «Добро пожаловать в Android Studio»:

Рисунок 9 - Добро пожаловать в Android Studio

Оно используется для запуска нового проекта Android Studio (start a new Android Studio project), работы с существующим проектом (Open an existing Android Studio project) и т. д.

Android Studio и создание первого проекта

Откроейте Android Studio. На начальном экране выберем пункт Start new Android Project (рисунок 9).

После этого отобразится диалоговое окно создания нового проекта:

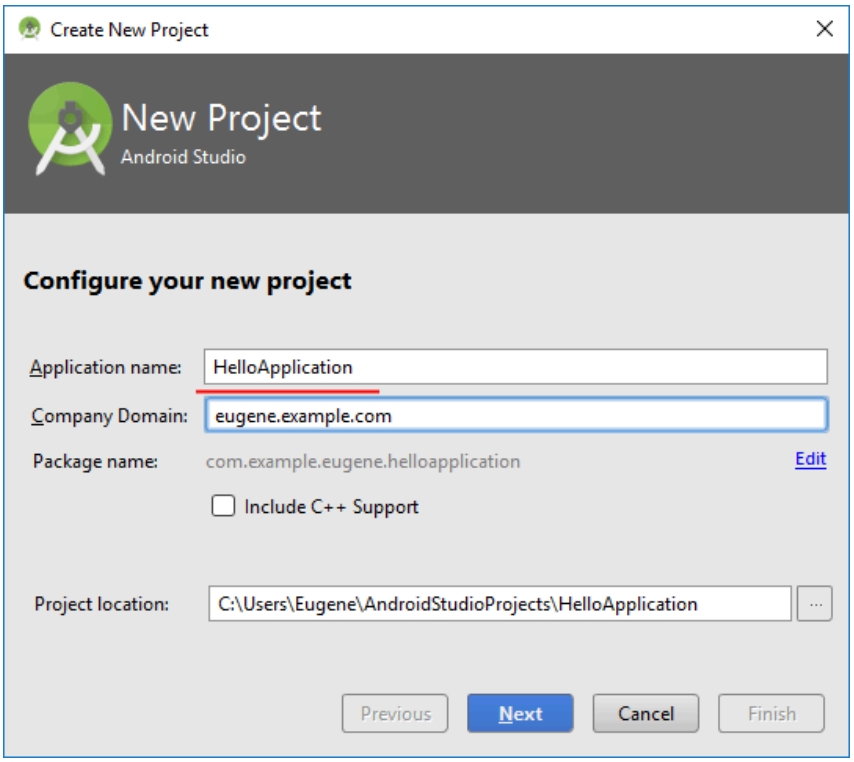


Рисунок 10 - Первый этап создания проекта

В окне создания нового проекта можно установить его начальные настройки:

- В поле Application Name вводится название приложения;
- В поле Company Domain указывается домен приложения или тот пакет классов, где будет размещаться главный класс приложения;
- В поле Project Location можно установить расположение файлов проекта на жестком диске.

Далее нажмите на кнопку Next. Следующий шаг:

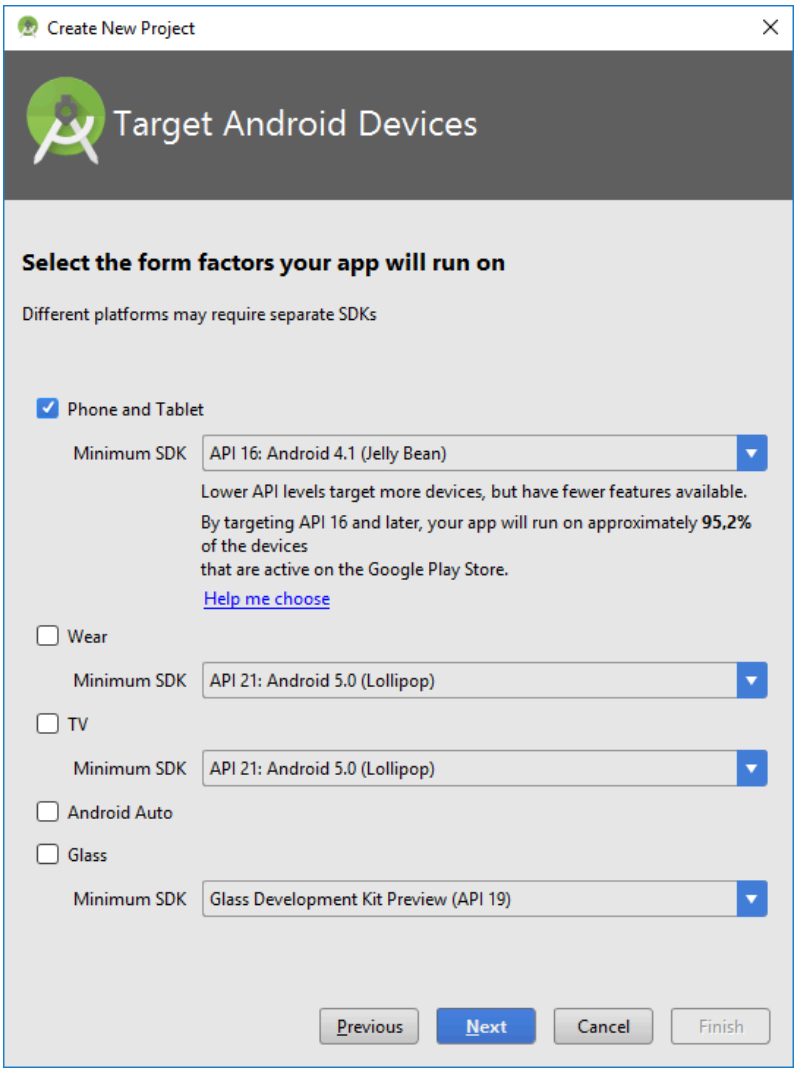


Рисунок 11 - Выбор минимальной поддерживаемой версии проекта

На этом шаге будет предложено установить минимальную поддерживаемую версию проекта. По умолчанию устанавливается версия Android 4.1, что покрывает более 95% устройств Android.

На следующем шаге надо выбрать шаблон проекта:

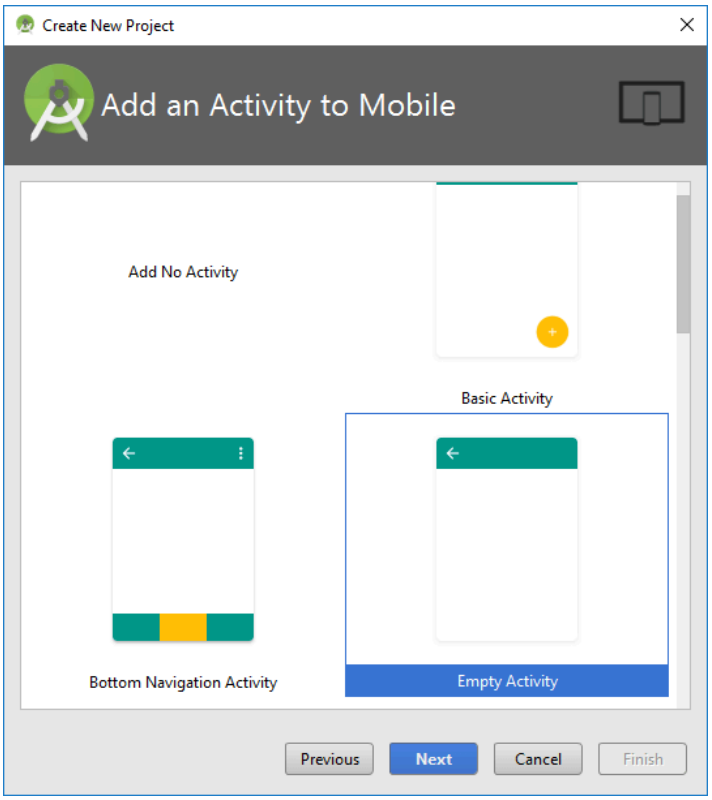


Рисунок 12 - шаблон проекта

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. Выберете шаблон Empty Activity.

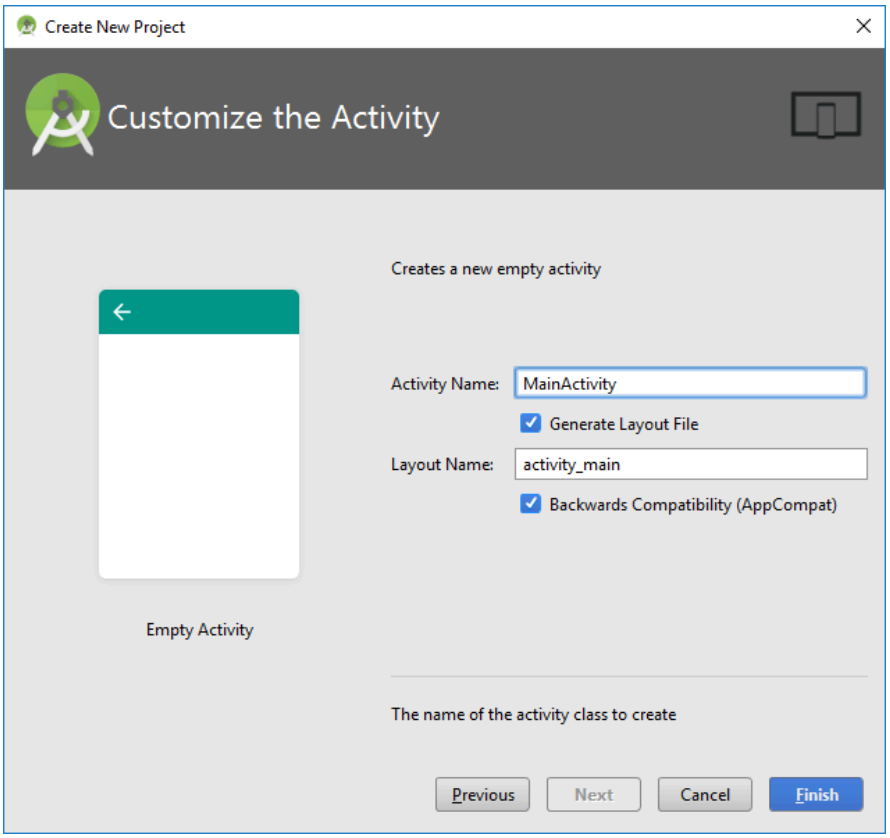


Рисунок 13 - Настройки проекта

При выборе Empty Activity на следующем шаге надо установить ряд настроек проекта:

- Activity Name: название главного класса приложения;
- Layout Name: название файла xml, в котором будет храниться определение визуального интерфейса;
- Generate Layout File: надо ли генерировать файл xml с определением визуального интерфейса.

Backwards Compatibility (AppCompat): в отмеченном состоянии позволяет установить обратную зависимость между различными версиями Android.

Нажав на кнопку Finish через некоторое время Android Studio создаст и откроет проект:

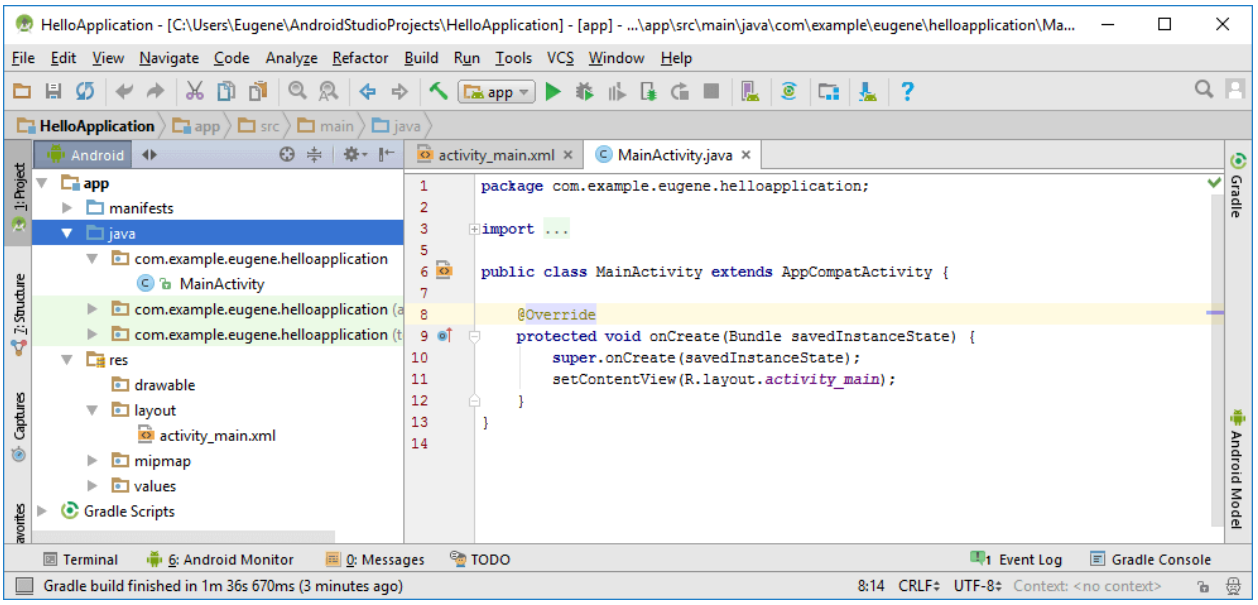


Рисунок 14 - Новый проект

Режим разработчика на телефоне. Для запуска и тестирования приложения можно использовать эмуляторы или реальные устройства. Но в идеале лучше тестировать на реальных устройствах.

Для использования мобильного устройства для тестирования на рабочую машину необходимо установить драйвер. Если смартфон от Google - Nexus 5/6/5x/6P или Google Pixel, то для его поддержки необходимо через SDK Manager установить пакет Google Usb Driver. Если же производитель аппарата - другой вендор, то надо установить тот USB-драйвер, который поставляется данным вендором. Если ОС - Windows 10, то там, как правило, система сама может найти через центр обновлений драйвер и установить его.

По умолчанию опции разработчика на смартфонах скрыты. Чтобы сделать их доступными, надо зайти в Settings > About phone (Настройки > О телефоне) и семь раз нажать Build Number (Номер сборки) (рисунок 15).

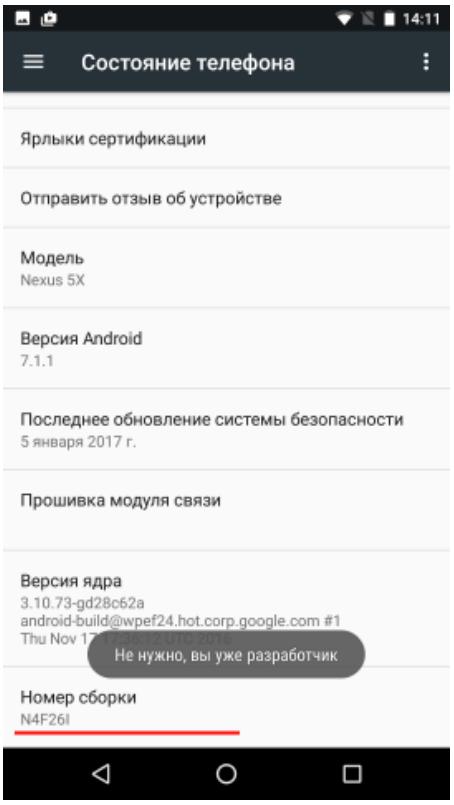


Рисунок 15 - Режим разработчика

Вернувшись назад вы увидите доступный пункт Developer options (Для разработчика). Перейдите к пункту Для разработчиков и включим возможность отладки по USB:

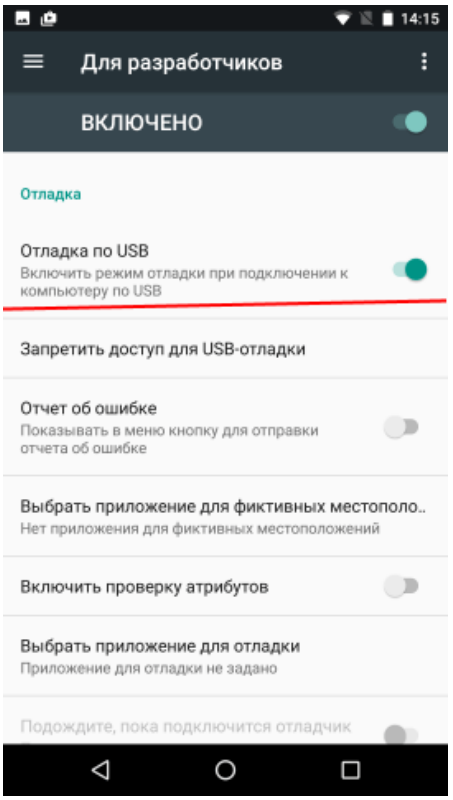


Рисунок 16 - Отладка по USB

Затем подключите устройство с ОС Android (если тестируете на реальном устройстве) и запустите проект, нажав на зеленую стрелочку на панели инструментов (рисунок 17).

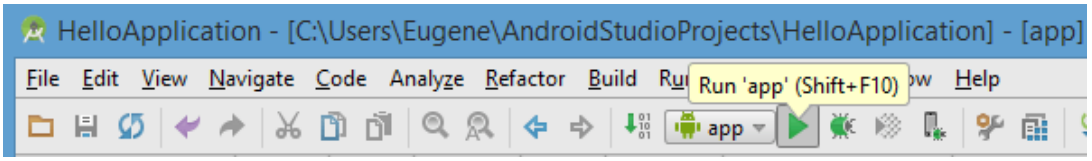


Рисунок 17 - Кнопка запуска проекта

Затем начнется построение проекта. Данный процесс может занять некоторое время, после чего отобразится диалоговое окно для выбора устройства для запуска. Здесь можно выбрать подключенный к компьютеру гаджет, либо эмулятор:

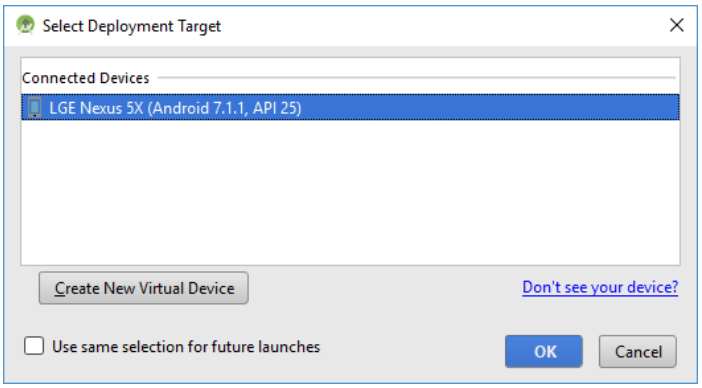


Рисунок 18 - Выбор устройства для запуска проекта

Выберем устройство и нажмите на кнопку OK. После запуска увидите приложение на экране устройства:

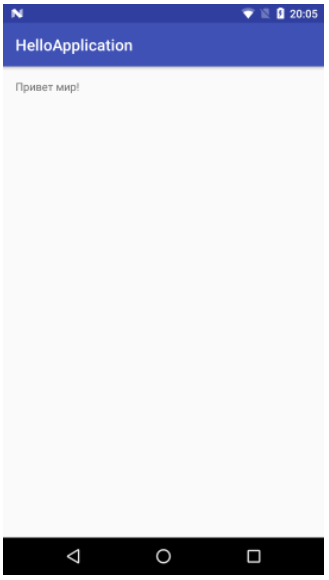


Рисунок 19 - Первое приложение на мобильном устройстве

Контрольные вопросы

1. почему при установке Android Studio требуется SDK?
2. почему лучше тестировать проект на реальном устройстве?

Лабораторная работа №11

Разработка калькулятора для ОС Android

Цель занятия

узнать:

- как создавать приложения для ОС Android;
- назначение файла activity_main.xml;
- как связаны поля класса с полями в файле activity_main.xml;
- как обрабатываются нажатия на кнопки.

научиться:

- создавать приложения для ОС Android;
- обрабатывать нажатия на кнопки;
- получать введенный пользователем текст;
- выводить результат на экран мобильного приложения.

Зная некоторые основы компоновки и такие элементы как TextView, EditText и Button, уже можно составить более-менее полноценное приложение.

Создайте новый проект и определите в файле activity_main.xml следующий интерфейс:

Рисунок 20 - Простейший интерфейс калькулятора

Корневой контейнер компоновки должен представлять элемент LinearLayout с вертикальной ориентацией. Первый элемент в нем - горизонтальный элемент LinearLayout, в котором должны быть определены два текстовых поля TextView: одно для вывода результата вычислений и одно для вывода текущего знака операции.

Затем должен идти элемент EditText, предназначенный для ввода чисел.

И далее расположите четыре элемента LinearLayout с горизонтальными рядами кнопок. Чтобы все кнопки занимали равное пространство внутри контейнера, для них установите атрибуты android:layout_weight="1" и android:layout_width="0dp".

Кроме того, для числовых кнопок в качестве обработчика нажатия установите метод onNumberClick, а для кнопок со знаками операций атрибут onClick указывает на метод onOperationClick.

Теперь класс MainActivity должен содержать следующий код:

```
package com.example.eugene.calculator;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
//текстовое поле для вывода результата
    TextView resultField;
//поле для ввода числа
    EditText numberField;
//текстовое поле для вывода знака операции
    TextView operationField;
// операнд операции
    Double operand = null;
// последняя операция
    String lastOperation = "=";

    @Override
    protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
   //метод,   получающий   все   поля   по   id   из
activity_main.xml
       setContentView(R.layout.activity_main);
        resultField                          =(TextView)
findViewById(R.id.resultField);
        numberField            =            (EditText)
findViewById(R.id.numberField);
        operationField          =           (TextView)
findViewById(R.id.operationField);
    }
    // сохранение состояния
    @Override
   protected void onSaveInstanceState(Bundle outState)
{
        outState.putString("OPERATION",
lastOperation);
        if(operand!=null)
            outState.putDouble("OPERAND", operand);
        super.onSaveInstanceState(outState);
    }
    // получение ранее сохраненного состояния
    @Override
```

```
    protected    void    onRestoreInstanceState(Bundle
savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceStat
e);
        lastOperation                                  =
savedInstanceState.getString("OPERATION");
        operand=
savedInstanceState.getDouble("OPERAND");
        resultField.setText(operand.toString());
        operationField.setText(lastOperation);
    }
    // обработка нажатия на числовую кнопку
    public void onNumberClick(View view){
        Button button = (Button)view;
        numberField.append(button.getText());
        if(lastOperation.equals("=")                   &&
operand!=null){
            operand = null;
        }
    }
    // обработка нажатия на кнопку операции
    public void onOperationClick(View view){
       Button button = (Button)view;
       String op = button.getText().toString();
                    String           number           =
numberField.getText().toString();
       // если введено что-нибудь
       if(number.length()>0){
            number = number.replace(',', '.');
            try{
               performOperation(Double.valueOf(number),
op);
            }catch (NumberFormatException ex){
                numberField.setText("");
            }
        }
        lastOperation = op;
        operationField.setText(lastOperation);
    }

    private   void   performOperation(Double   number,
String operation){
// если операнд ранее не был установлен (при вводе
самой первой операции)
        if(operand ==null){
```

```
            operand = number;
        }
        else{
            if(lastOperation.equals("=")){
                lastOperation = operation;
            }
            switch(lastOperation){
                case "=":
                    operand =number;
                    break;
                case "/":
                    if(number==0){
                        operand =0.0;
                    }
                    else{
                        operand /=number;
                    }
                    break;
                case "*":
                    operand *=number;
                    break;
                case "+":
                    operand +=number;
                    break;
                case "-":
                    operand -=number;
                    break;
            }
        }
     resultField.setText(operand.toString().replace('.
', ','));
        numberField.setText("");
    }
}
```

Подробнее об этом классе. Вначале в методе onCreate() происходит получение всех полей из activity_main.xml, текст которых будет изменяться:3

```
resultField                                    =(TextView)
findViewById(R.id.resultField);
numberField                  =                  (EditText)
findViewById(R.id.numberField);
operationField = (TextView) findViewById(R.id.operatio
nField);
```

Результат операции будет попадать в переменную operand, которая представляет тип Double, а знак операции - в переменную lastOperation:

```
Double operand = null;
String lastOperation = "=";
```

Так как при переходе от портретной ориентации к альбомной или наоборот можно потерять все введенные данные, чтобы их не потерять, их нужно сохранить в методе onSaveInstanceState() и обратно получить в методе onRestoreInstanceState().

При нажатии на числовую кнопку будет вызываться метод onNumberClick, в котором добавляется введенная цифра или знак запятой к тексту в поле numberField:

```
Button button = (Button)view;
numberField.append(button.getText());
 if(lastOperation.equals("=") && operand!=null){
    operand = null;
}
```

При этом если последняя операция представляла собой получение результата (знак "равно"), тогда сбрасывается переменная operand.

В методе onOperationClick происходит обработка нажатия на кнопку со знаком операции:

```
Button button = (Button)view;
String op = button.getText().toString();
String number = numberField.getText().toString();
if(number.length()>0){
    number = number.replace(',', '.');
    try{
          performOperation(Double.valueOf(number),
op);
    }catch (NumberFormatException ex){
        numberField.setText("");
    }
}
lastOperation = op;
operationField.setText(lastOperation);
```

Здесь происходит получение ранее введенного числа и введенную операцию и передача их в метод performOperation(). Так как в метод

передается не просто строка, а число Double, то нужно преобразовать строку в число. И поскольку теоретически могут быть введены нечисловые символы, то для отлова исключения, которое может возникнуть при преобразовании используется конструкция try...catch.

Кроме того, так как разделителем целой и дробной части в Double в java является точка, то надо заменить запятую на точку, так как предполагается, что используется в качестве разделителя запятая.

А методе performOperation() выполняется собственно операция. При вводе первой операции, когда операнд еще не установлен, просто устанавливается операнд:

```
if(operand ==null){
    operand = number;
}
```

При вводе второй и последующих операций применяется предыдущая операция, знак которой хранится в переменной lastOperation, к операнду operand и второму числу, которое было введено в числовое поле. Полученный результат операции сохраняется в переменной operand.

Задание для самостоятельной работы:

1. Задайте фоновый цвет кнопок.
2. Увеличьте размер текста на кнопках и в поле EditText.
3. Добавьте кнопки и операции возведения в квадрат и в любую другую степень.

Контрольные вопросы

3. для чего предназначен файл activity_main.xml?
4. как обратиться ко всем полям файла activity_main.xml?
5. в каких единицах измеряется размер шрифта?
6. какой атрибут задает цвет фона у кнопки, текста?

Лабораторная работа №12

Разработка программ для ОС Android

Цель занятия

узнать:

- о работе с изображениями;
- о вводе и выводе значений на экран мобильного приложения.

научиться:

- разрабатывать Android-приложения;
- вводить и получать данные через компонент EditText;
- обрабатывать нажатие на кнопку.

Задания

1 задание Создать приложение, которое будет лайкать ваше фото и отображать количество лайков.

!!! Вариант согласно № п.п. (см. журнал) !!!

Вариант	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Задание	а	б	в	г	д	е	ж	а	б	в	г	д	е	ж

2 задание Согласно варианту создать приложение, которое позволит рассчитать количество грамм в чайных ложках, в столовых ложках, в стаканах и наоборот для:

1) Вода;
2) Сахар;
3) Соль;
4) Манка;
5) Крахмал;
6) Растительное масло;
7) Мука.

Контрольные вопросы

1. для чего предназначен файл activity_main.xml?
2. как обратиться ко всем полям файла activity_main.xml?
3. в каких единицах измеряется размер шрифта?
4. какой атрибут задает цвет фона у кнопки, текста?

Контрольная работа

Задания

!!! Вариант согласно № п.п. (см. журнал) !!!

Вариант	1	2	3	4	5	6	7	8	9	10	11	12
Задание	1	2	3	4	5	6	1	2	3	4	5	6

Создайте полноценное Android-приложение согласно номеру варианта.

1. Журнал с оценками по 3-м предметам. Условие: необходимо выбрать ФИО студента (должна быть строка с вводом текста) с оценками по всем предметам.
2. Калькулятор для перевода различных систем счисления (двоичная, восьмеричная, десятичная, шестнадцатеричная).
3. Программа для перевода миллиметров, сантиметров, дециметров, метров в каждую из перечисленных.
4. Стандартный календарь, показывающий текущую дату.
5. MP3-плеер с play-листом, функциями перемотки на одну песню, паузой и воспроизведением.
6. Таймер.

Контрольные вопросы

1. какие атрибуты и элементы были выбраны для создания мобильного приложения?
2. что такое Activity?
3. сколько Activity в вашем приложении, сколько классов?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А. Н. Семочкин Язык программирования Java / А. Н. Семочкин, издательство БГПУ, 2006 г, 79 с.
2. Д. Гослинг, К. Арнольд Язык программирования Java / Д. Гослинг, К. Арнольд, издательский дом "Питер", 2002 г, 50 с.
3. А. В. Мезенцев Основы языка программирования Java / А. В. Мезенцев, издательство ИГУ, 2012г, 125 с.
4. О.Л. Стесик, М.М. Степанова Основы программирования. Курс на базе языка программирования java / О.Л. Стесик, М.М. Степанова, СПб, 2016 г, 62 с.
5. Общие сведения о платформе Android [Электрон. ресурс]. – Режим доступа: https://developer.android.com/guide.
6. Сайт о программировании [Электрон. ресурс]. – Режим доступа: https://metanit.com/java/android.
7. Учебник по программированию под Android [Электрон. ресурс]. – Режим доступа: http://elsof.ru/android/tutorial/.
8. Start Android – учебник по Android для начинающих и продвинутых [Электрон. ресурс]. – Режим доступа: http://startandroid.ru/ru/.