Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ И ДИЗАЙНА»

Кафедра информационных технологий

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В ИНЖЕНЕРНОЙ ПРАКТИКЕ

Методические указания и контрольные задания
для студентов направления подготовки 09.03.03
заочной формы обучения

Составители:
И. А. Николаев
Е. С. Кокорин

Санкт-Петербург
2016

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

По своей производительности мобильные устройства под управлением различных платформ уже достигли уровня персональных компьютеров.

Разработка приложений и распространение стороннего программного обеспечения как отдельное направление мобильных технологий в последнее время приобрело значительную актуальность. При традиционном подходе все программное обеспечение уже входило в состав предустановленной операционной системы. К нему предъявлялись жесткие требования – программы должны были быть полностью оптимизированными под конкретную операционную систему и проходить различные уровни тестирования: *GUI*, стресс-тестирование, “*smoke*” *test* и т. д.

С постепенным развитием информационных технологий и появлением новых мобильных устройств под управлением различных операционных систем, стало развиваться такое направление, как разработка мобильных приложений. Таким образом, у сторонних разработчиков появилась возможность создавать собственные приложения для различных операционных систем и опубликовывать их, либо делиться исходными файлами друг с другом, не публикуя приложение.

Разработка мобильных приложений во многих аспектах сходна с разработкой приложений для настольных компьютеров. В то же время разработка мобильных приложений ставит перед программистами новые задачи. К ним, в частности, относятся оптимальное управление памятью мобильной платформы, учет ограниченного заряда батареи, а также учет возможной нестабильности беспроводного сетевого соединения.

В указаниях рассматриваются основы программирования на языке Java для приобретения базовых навыков разработки простых мобильных приложений.

Контрольная работа заключается в выполнении семи заданий. Каждое задание включает в себя два теоретических вопроса и две практических задачи. Приведены примеры выполнения заданий: краткие ответы на вопросы и фрагменты программного кода для первой задачи. На проверку по каждому заданию необходимо прислать подробные ответы на вопросы и отлаженные тексты программ.

1. ЗАДАНИЕ. ТИПЫ ДАННЫХ И ПЕРЕМЕННЫЕ

1.1. Теоретический материал

1.1.1. Простые типы данных

Java является строго типизированным языком программирования. Это означает, что переменная описывается строго определенным типом данных. В *табл. 1* рассмотрены простые типы данных, интегрированные в Java.

Таблица 1. Простые типы данных

Тип данных	Разрядность, бит	Занимаемый объем памяти, байт	Диапазон допустимых значений	Краткое описание
byte	8	1	–128 до 127	8-разрядное целое число
short	16	2	--32.768 до 32.767	Короткое целое число
int	32	4	–2.147.483.648 до 2.147.483.647	Целое число
long	64	8	–9.223.372.036.854.775.808 до 9.223.372.036.854.775.807	Длинное целое число
float	32	4	–3.4*10^38 до 3.4*10^38	Хранит число с плавающей точкой одинарной точности

Окончание табл. 1

Тип данных	Разрядность, бит	Занимаемый объем памяти, байт	Диапазон допустимых значений	Краткое описание
double	64	8	– +4.9*10^324 до + –1.8*10^308	Хранит число с плавающей точкой двойной точности
boolean	-	-	true/ false	Представляет логические значения
char	16	2	0 до 65536	Хранит одиночный символ в кодировке Unicode

Примечательно, что первые четыре типа данных (*byte, short, int и long*) – это целочисленные типы данных, т. е. они хранят *целые числа*.

1.1.2. Объявление переменных

Переменные объявляются следующим образом: сперва идет *тип_данных*, затем *имя_переменной*. Например, *int x, int y, double z* и т. д.

В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

– имя может содержать любые алфавитно-цифровые символы, а также знак подчеркивания, при этом первый символ в имени не должен быть цифрой;

– в имени не должно быть знаков пунктуации и пробелов;

– имя не может быть ключевым (зарезервированным) словом языка Java.

Помимо этого, Java является регистрозависимым языком. Поэтому, если объявить переменную как *int rest* и *int REST*, то это будут два разные переменные.

Объявив переменную, мы можем тут же присвоить ее значение или инициализировать ее. Инициализация переменных представляет присвоение переменной начального значения, например: *int x=5*. Знак равенства, в данном случае, читается как “Икс имеет значение 5”. Если мы не проинициализируем переменную до ее использования, то мы можем получить ошибку, например, в следующем случае:

```
int test;
System.out.print(test);
```

В данном случае переменная *test* не инициализирована и, соответственно, никакое значение переменной не будет выведено на экран.

1.1.3. Объявление констант

Значение всех переменных можно менять сколько угодно раз, по мере необходимости. Чтобы избежать возможных нежелательных изменений, прибегают к объявлению констант. Константы в Java объявляются с помощью ключевого слова *final*. Пример:

```
final int z = 5;
```

После этого, мы уже не сможем изменить значение переменной *z* в другом участке кода, т.к. z – константа.

1.1.4. Работа с символами и строками

В качестве значения переменная символьного типа получает одиночный символ, заключенный в *одинарные кавычки*: *char m='r'*. Кроме того, переменной символьного типа также можно присвоить целочисленное значение от *0* до *65536*. В этом случае переменная опять же будет хранить символ, а целочисленное значение будет указывать на номер символа в таблице символов Unicode.

Одинарные и двойные кавычки имеют существенные отличия. Нельзя заключить переменную символьного типа в двойные кавычки, так как это уже будет переменная строкового типа *String*. Грубо говоря, *'r'* не идентично *"r"*.

Переменные строкового типа не относятся к примитивным (простым) типам данных.

1.2 Вопросы и задачи к заданию 1

1. Какие типы данных вам знакомы? Перечислите их.

2. Дайте определение термину "Инкапсуляция" и приведите примеры.

3. Решите задачу: «Вы стоите в поле и, вдруг, увидели вспышку молнии. Через 8 с раздался раскат грома. Необходимо написать программу, которая рассчитывает расстояние от вашего местоположения до грозы».

4. Определить ваш вес на Марсе, зная что ускорение силы тяжести на нем примерно 3,711 м/с^2.

1.3. Пример выполнения задания 1

1. Типы данных бывают:

– *integer* (целочисленные) – хранит целое число в своем диапазоне и занимает 4 байта;

– *float* – хранит число с плавающей точкой одинарной точности в своем диапазоне и занимает 4 байта;

– *double* - хранит число с плавающей точкой одинарной точности в своем диапазоне и занимает 8 байт;

– *byte* - хранит целое число в своем диапазоне и занимает 1 байт;

– *short* - хранит целое число в своем диапазоне и занимает 2 байта;

– *long*- хранит целое число в своем диапазоне и занимает 8 байт;

– *char* – хранит одиночный символ в кодировке Unicode и занимает 2 байта;

– *boolean* – хранит значение *true* или *false*.

– *string* – работает со строками.

2. *Инкапсуляция* – это базовый принцип объектно-ориентированного программирования (ООП), который позволяет объединить данные и код, который их обрабатывает, тем самым, скрыть реализацию от пользователя. Данный принцип гарантирует нам сохранность данных от нежелательных изменений.

Возьмем, к примеру, автомобиль. В данном случае – это объект. Он включает в себя множество функций и операций (начиная переключением передач и заканчивая регулировкой сидений), которые, фактически, объединены в одно целое – в автомобиль. Когда мы сидим за рулем автомобиля и переключаем передачи, мы знаем, что произойдет, когда мы включим 1 передачу, затем 2 и т. д. Мы знаем конечный результат. Но сама реализация данного метода находится “глубже” – мы не видим, как колеса “разъединяются” с двигателем, как воздействуют друг с другом различные механизмы, чтобы был получен конечный, известный для нас результат. Это скрыто и защищено от нас, мы не может это “изменить” (если не разбирать автомобиль). Также при переключении передач у нас самопроизвольно не выключается магнитола или не включается дальний свет. Данный метод также защищен от воздействия других, лишних методов. В этом и заключается основной принцип инкапсуляции.

3. Условие:

“Вы стоите в поле и вдруг увидели вспышку молнии. Через 8 с раздался раскат грома. Необходимо написать программу, которая рассчитывает расстояние от вашего местоположения до грозы”.

Решение:

Пусть t =8 с, а скорость звука в воздухе V =340.29 м/с. Тогда, зная время и скорость, найдем расстояние S по формуле $S = V * t$.

Реализуем данную задачу программным методом.

Для начала откроем главный класс, назовем его *Rest_1*.

```
publicclassRest_1 { //создаем класс
}
```

Следующим шагом создадим метод, который будет служить точкой входа в программу для компилятора.

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

Далее определимся с типами данных и переменными, которые мы будем использовать.

Обозначим время как *int t* и передадим ей значение 8 с. Для скорости определим тип *double* – так как скорость задана с точностью до сотых, чтобы обеспечитьточность расчетов, используем соответствующий тип данных.

Также необходимо определить тип данных и выделить переменную под дистанцию. Определим ее как *double* (см. выше) и дадим ей имя *dist*.

```
int t = 8;
double v = 340.29, dist;
```

Зададим в программе ее вычисление по формуле.

```
dist = v * t;
```

И выведем полученный результат на экран, с помощью соответствующего метода.

```
System.out.println (“Расстояние равно = ” + dist + “ метров”);
```

Не забываете конкатенировать значения и закрывать методы и классы.

Ниже полностью приведен исходный код программы.

```
public class Rest_1 {//создаем класс
public static void main(String[] args) {
int t = 8;//инициализируем переменную
double v = 340.29, dist;
dist = v * t;
System.out.println ("Расстояние равно = " + dist + " метров"); //Выводим результат на экран
}
}
```

Результатом данной программы будет: *“Расстояние равно = 2722.32 м”*.

2. ЗАДАНИЕ 2. ОПЕРАЦИИ ЯЗЫКА JAVA

2.1. Теоретический материал

2.1.1. Арифметические операции

Операции языка Java во многом идентичны операциям в других языках программирования. По типу они подразделяются на следующие:

– унарные – операции с одним операндом;

– бинарные – операции с двумя операндами;

– тернарные – операции с тремя операндами.

Синтаксис основных арифметических операций приведен в *табл. 2*.

Таблица 2. Арифметические операции

Операция	Пример операции	Описание
+	z = x + y	Сложение
–	z = x - y	Вычитание
*	z = x * y	Умножение
/	z = x / y	Деление
%	z = x % y	Остаток от деления
++	z = ++x Префиксный инкремент	Увеличение переменной на единицу; сначала переменная x увеличивается на единицу, а затем присваивается z
	z =x++ Постфиксный инкремент	Увеличение переменной на единицу; сначала переменная x присваивается z, а затем увеличивается на единицу.
– –	z = --x Префиксный декремент	Обратно префиксному инкременту
	z = x-- Постфиксный	Обратно постфиксному

	декремент	инкременту

2.1.2. Операции сравнения (числовые значения)

Таблица 3. Числовые значения

Оператор	Пример операции	Описание
= =	z = x = = y	Переменная z имеет значение true, если x и y равны, иначе false
!=	z = x != y	Переменная z имеет значение true, если x и y не равны, иначе false
<	z = x<y	Переменная z имеет значение true, если x меньше y, иначе false
>	z = x>y	Переменная z имеет значение true, если x больше y, иначе false
<=	z = x<= y	Переменная z имеет значение true, если x меньше или равно y, иначе false
>=	z = x>= y	Переменная z имеет значение true, если x больше или равно y, иначе false

2.1.3. Операции сравнения (булевы значения)

Таблица 4. Булевы значения

Оператор	Пример операции	Описание
|	z = x | y	Переменная z имеет значение true, если x или y, или они одновременно имеют значение true, иначе false

Окончание табл. 4

Оператор	Пример операции	Описание
&	z = x & y	Переменная z имеет значение true, если x и y одновременно имеют значение true, иначе false
^	z = x ^ y	Переменная z имеет значение true, если x или y (но не одновременно) имеют значение true, иначе false
!	z != x	Переменная z имеет значение true, если x имеет значение false, иначе false
||	z = x || y	Переменная z имеет значение true, если x или y, или они одновременно имеют значение true, иначе false
&&	z = x && y	Переменная z имеет значение true, если x и y одновременно имеют значение true, иначе false

Представленные операции *z = x | y и z = x || y (z = x&y и z = x&&y)* выполняют одни и те же действия, приводят к одному и тому же результату, но имеют совершенно разный подход.

Исходя из условия, операция *z = x | y* передаст переменной *z* значение *true*, если *x* или *y*, или они одновременно имеют значение *true*, в противном случае *false*. Данная операция проверит *каждый* операнд и выдаст конечный результат.

А операция *z = x || y* проверит до первого удовлетворяющего (или неудовлетворяющего значения). Сперва вычисляется значение *x*, и если оно имеется значение *true*, то вычисление *y* уже не имеет смысла, исходя из условия, и переменной *z* будет передано значение *true*.

Аналогичный принцип и для операций *z = x&y и z = x&&y*.

2.1.4 Операции присваивания

Таблица 5. Операции присвоения

Оператор	Пример операции	Описание
+=	z += x	Переменной z присваивается значение сложения z и x
-=	z -= x	Переменной z присваивается значение вычитания z и x
*=	z *= x	Переменной z присваивается значение произведения z и x
/=	z /= x	Переменной z присваивается значение деления z и x
%=	z %= x	Переменной z присваивается остаток от деления z на x

2.2. Вопросы и задачи к заданию 2

1. Одинаковы ли следующие логические операции: *c = a | b* и *c = a || b*? Ответ обоснуйте.

2. Что выведет следующая программа? Дайте пояснения к своему ответу.

```
public class Main {
```

```
public static void main(String[] args) {
        String s = "ОДИН" + 1 + 14 + "ДВА" + 4 +"ТРИ"
+ 7 + 3 + "ЧЕТЫРЕ" + "ПЯТЬ" + 6 + 3 + 2 + "ШЕСТЬ";
System.out.println (s);
    }
}
```

3. Напишите программу, которая вычисляет гипотенузу прямоугольного треугольника по двум катетам. Необходимо разрешить пользователю самостоятельный ввод значений.

4. В переменной *zh* хранится натуральное двузначное число. Вычислите и выведите на экран сумму цифр числа *zh*.

2.3 Пример выполнения задания №2:

1. Для того, чтобы сравнивать данные логические операции, необходимо понимать их реализацию.

Итак, операция *c = a | b* имеет следующее значение – *c* принимает значение *true*, если либо *a* имеет значение *true*, либо *b* имеет значение *true*, либо они *одновременно* имеют значение *true.*

Операция *c = a || b* идентична операции *c = a | b,* за исключением одного важного момента.

В операции *c = a || b, c* принимает значение *true* до первого удовлетворяющего условия. Например, если *a* имеет значение *true*, то *c* автоматически принимает значение *true,* в соответствии с условием.

В операции *c = a |* проверяются все значения, несмотря на удовлетворяющие условия.

Такой подход позволяет экономить ресурсы и, тем самым, увеличивать производительность.

2. В данном примере результатом выполнения программы будет следующее: “ОДИН114ДВА4ТРИ73ЧЕТЫРЕПЯТЬ632ШЕСТЬ”.

Тип данных объявлен как *String* (строковый тип данных), значит мы работаем со строками и математической реализации в данном примере не произойдет. Т.е. у нас в ответе не будет суммы, например, *6 + 3 + 2*, числа просто конкатенируются и образуют цепочку символов *632*. Исходя из этого, получаем соответствующий ответ.

3. *Условие:* “Напишите программу, которая вычисляет гипотенузу прямоугольного треугольника по двум катетам. Необходимо разрешить пользователю самостоятельный ввод значений в программу”.

Решение:

Для начала необходимо разобраться с тем, что мы будем реализовывать. Известно, что гипотенуза определяется по следующему правилу: “*Квадрат длины гипотенузы равен сумме квадратов длин катетов*”. Пусть ***a***, *b*– катеты, а *c* – гипотенуза.

Реализуем данное правило в программе. Сперва импортируем метод *Scanner* из библиотеки *util* для того, чтобы реализовать функцию считывания данных с клавиатуры.

```
import java.util.Scanner;
Затем, создадим класс с именем, например, Pifagor и
сразу же создадим метод, который будет служить точкой
входа в программу для компилятора.
public class Pifagor {
public static void main(String[] args) {

}
}
```

Следующим шагом выделим память под объект *Scanner*, который использует метод *System.in* (поток на вход).

```
Scanner in = new Scanner (System.in);
```

Далее выведем информативное сообщение для пользователя с предложением ввести значения катетов по средствам *System.out* (поток вывода) и считываем все введенные значения с помощью метода *in.nextDouble();* (*double*, потому что мы используем соответствующий тип данных; к примеру, если бы мы использовали *int*, то было бы *in.nextInt();*).

```
System.out.println ("Введите значение длины первого катета");
    double a = in.nextDouble();
System.out.println ("Введите значение длины второго катета");
    double b = in.nextDouble();
```

Теперь можно найти значение гипотенузы. Для того чтобы возвести число в степень, мы используем функцию *Math.pow (x, y);*, где *x* – число, которое мы должны возвести в степень, а *y*–степень, в которую необходимо возвести число.

Для квадратного корня существует функция *Math.sqrt ();*, где в круглых скобках указывается значение, из которого нужно извлечь квадратный корень.

```
double c = Math.pow(a, 2) + Math.pow(b, 2);
System.out.println("Гипотенуза равна = " + Math.sqrt (c));
```

Ниже полностью приведен исходный код программы.

```
import java.util.Scanner;//подключаем библиотеку
public class Pifagor {
public static void main(String[] args) {
        Scanner in = new Scanner (System.in);
System.out.println ("Введите значение длины первого катета");
double a = in.nextDouble();//считываем значение a
System.out.println ("Введите значение длины второго катета");
double b = in.nextDouble();//считываем значение b
double c = Math.pow(a, 2) + Math.pow(b, 2);
System.out.println("Гипотенуза равна = " + Math.sqrt (c));
}
}
```

Данную программу можно упростить, к примеру, можно не выделять отдельную переменную для гипотенузы и не использовать функцию *Math.pow (x, y)*;.

```
import java.util.Scanner;
public class Pifagor {
public static void main(String[] args) {
    Scanner in = new Scanner (System.in);
    System.out.println ("Введите значение длины первого катета");
    double a = in.nextDouble();
```

```
        System.out.println ("Введите значение длины
    второго катета");
        double b = in.nextDouble();
        System.out.println("Гипотенуза равна = " +
    Math.sqrt (a*a + b*b));
    }
    }
```

Исходя из определения, что квадрат числа – это исходное число, умноженное само на себя, имеем право записать *a*a*, что никак не повлияет на результат выполнения программы.

3. ЗАДАНИЕ 3. ПАКЕТЫ И УСЛОВНЫЕ КОНСТРУКЦИИ

3.1. Теоретический материал

3.1.1. Конструкция if/else

Условные конструкции – важный инструментарий языка Java. Условные конструкции позволяют направить программу по определенному пути развития, в зависимости от условий.

Условные конструкции бывают двух типов:

– if/else;

– switch/case.

Данные конструкции идентичны друг другу, за одним исключением – в отличие от конструкции *if/else*, конструкция *switch/case* может обрабатывать сразу несколько условий.

Рассмотрим первую условную конструкцию *if/else*. Она имеет следующий синтаксис:

```
if(условие){
```

```
        код, выполняющийся при удовлетворении
условия;
}
else{
«запасной выход»;
```

Если условие, которое содержит в себе блок *if*, истинно, то выполняется код, который хранит данный блок. В этом случае, блок *else* не выполняется! Равно и обратное. Пример:

```
int chn = 2;
int chz = 5;
if(chn>chz){
     System.out.println(chn + “больше, чем” + chz);
}
else if(chn<chz){
     System.out.println(chn + “меньше, меньше” +
chz);
}
else{
      System.out.println(chn + “и” + chz + “равны”);
}
```

В данном примере, мы рассматриваем сравнение двух чисел между собой. Вначале присваиваем значения переменным и задаем различные пути развития с помощью *if/else*.

Обратите внимание, что здесь мы использовали оператор *else if*. Он может применяться, если у нас существует более двух условий.

Хотя для обработки нескольких условий иногда целесообразнее использовать конструкцию *switch/case*.

3.1.2. Конструкция switch/case

Конструкция *switch/case*, как уже говорилось ранее, предназначена для обработки сразу нескольких условий. Пример:

```
int chn = 2;
switch(chn){
     case 4:
          System.out.println("Ответ#1");
          break;
     case 5:
        System.out.println("Ответ#2");
        num++;
        break;
     case 2:
          System.out.println("Ответ#3");
        break;
 case 3:
        System.out.println("Ответ#4");
        break;
     default:
        System.out.println("Ничего отвечать");
}
```

В данном примере после *switch* идет сравниваемое значение. Значение последовательно сравнивается со значениями блока *case*. Если есть совпадение, то выполняется код, помещенный в соответствующий блок *case*.

3.2. Вопросы и задачи к заданию 3

1. Чем JRE отличается от JDK?

2. Что такое тернарная операция?

3. Напишите программу, которая решает квадратные уравнения вида:

$$ax^2 + bx + c = 0.$$

При отрицательном дискриминанте решением можно пренебречь и вывести соответствующее сообщение. Необходимо разрешить пользователю вводить свои значения с клавиатуры.

4. Написать программу, сообщающую на экран, является ли число, записанное в переменную zh, целым или нет.

3.3 Пример выполнения задания 3

1. *JRE (Java Runtime Environment)*– среда для запуска приложений, написанных на языке программирования Java.

JDK (Java Development Kit)– инструменты для разработки приложений, написанных на языке программирования Java. JDK включает в себя JRE.

2. Тернарная операция включает в себя три операнда. Она имеет следующий вид:

[первый операнд – условие] ?[второй операнд] : [третий операнд]

В зависимости от условия операция возвращает или *второй*, или *третий* операнд: если условие имеет значение *true*, то возвращается *второй* операнд, в противном случае возвращается *третий* операнд.

3. *Условие.* “Напишите программу, которая решает квадратные уравнения вида

ax^2 + bx + c = 0.

При отрицательном дискриминанте решением можно пренебречь и вывести соответствующее сообщение. Необходимо разрешить пользователю вводить свои значения с клавиатуры”.

Решение:

При решении квадратных уравнений вида: ax^2 + bx + c = 0 могут возникнуть три состояния:

a. Дискриминант больше нуля.

В таком случае, корни квадратного уравнения определяются по следующей формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

b. Дискриминант равен нулю.

В этом случае имеется два одинаковых вещественных корня квадратного уравнения, которые определяются по аналогичной формуле, указанной выше.

c. Дискриминант меньше нуля.

Если дискриминант меньше нуля, значит, квадратное уравнение не имеет вещественных корней и они комплексные. Так как в условии сказано, что данным путем развития можно пренебречь, для него выведем соответствующее информативное сообщение.

Сперва, импортируем метод *Scanner* из библиотеки *util* для того, чтобы реализовать функцию считывания данных с клавиатуры.

```
import java.util.Scanner;
```

Затем создадим класс с именем *RKU* и метод, который будет служить точкой входа в программу для компилятора, выделим память под объект *Scanner*, который использует метод *System.in* (поток на вход) и инициализируем переменные.

```
public class RKU {
    public static void main(String[] args) {
Scanner in = new Scanner(System.in);
int a, b, c, d;
double x1, x2;
}
```

Далее выведем информационное сообщение о формате решаемых уравнений и сообщение для пользователя с предложением ввести значения коэффициентов уравнения посредством *System.out* (поток вывода) и считываем все введенные значения с помощью метода *in.nextInt();*

```
System.out.println("Данная программа решает уравнения
вида: ax^2 + bx + c = 0");
System.out.println("Введите a: ");
a = in.nextInt();
System.out.println("Введите b: ");
        b = in.nextInt();
System.out.println("Введите c: ");
c = in.nextInt();
System.out.println("Ваше уравнение имеет вид: " +'\n'
+ a + "x^2" + "+" + b + "x" + "+" + c + "+" + " =0");
```

В конце мы вывели на экран уравнение, которое ввел пользователь. Определим формулы расчета дискриминанта и корней квадратного уравнения в зависимости от значения дискриминанта.

```
    d = (int) (Math.pow(b, 2) - (4 * a * c));
    System.out.println("Дискриминант равен = " + d);
    if (d > 0){
    x1 = ((-1) * b + Math.sqrt(d)) / 2 * a;
    x2 = ((-1) * b - Math.sqrt(d)) / 2 * a;
    System.out.println("Корни квадратного уравнения: " +
'\n' + "x1 = " + x1 + '\n' + "x2 = " + x2);
    }
    else if (d == 0){
    x1 = ((-1) * b + Math.sqrt(d)) / 2 * a;
    x2 = ((-1) * b - Math.sqrt(d)) / 2 * a;
    System.out.println("Так как d = 0, у нас имеется два
одинаковых  вещественных корня уравнения: " + '\n' + "x1
= " + x1 + '\n' + "x2 = " + x2);
            }
    else
    System.out.println("Квадратное  уравнения  не  имеет
вещественных корней");
```

Полный исходный код данной программы приведен ниже.

```
    import java.util.Scanner;
    public class RKU {
    public static void main(String[] args) {
           Scanner in = new Scanner(System.in);
    int a, b, c, d;
```

```
    double x1, x2;
    System.out.println("Данная программа решает уравнения вида: ax^2 + bx + c = 0");

    System.out.println("Введите a: ");
    a = in.nextInt();
    System.out.println("Введите b: ");
            b = in.nextInt();
    System.out.println("Введите c: ");
            c = in.nextInt();
    System.out.println("Ваше уравнение имеет вид: " +'\n' + a + "x^2" + "+" + b + "x" + "+" + c + "+" + " =0");

            d = (int) (Math.pow(b, 2) - (4 * a * c));
    System.out.println("Дискриминант равен = " + d);

    if (d> 0){ //если дискриминант больше нуля
    x1 = ((-1) * b + Math.sqrt(d)) / 2 * a;
    x2 = ((-1) * b - Math.sqrt(d)) / 2 * a;
    System.out.println("Корни квадратного уравнения: " + '\n' + "x1 = " + x1 + '\n' + "x2 = " + x2);
    }

    else if (d == 0){//если равен нулю
    x1 = ((-1) * b + Math.sqrt(d)) / 2 * a;
    x2 = ((-1) * b - Math.sqrt(d)) / 2 * a;
    System.out.println("Так как d = 0, у нас имеется два одинаковых  вещественных корня уравнения: " + '\n' + "x1 = " + x1 + '\n' + "x2 = " + x2);// ‘\n’ – оператор переноса строки
```

```
            }
    else
    System.out.println("Квадратное уравнения не имеет
вещественных корней");
        }
    }
```

4. ЗАДАНИЕ 4. ЦИКЛЫ

4.1. Теоретическая часть

4.1.1. Цикл for

Формально, цикл for выполняет роль счетчика. Задается цикл следующим образом:

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

Как мы видим, цикл поделен на три составляющие: *инициализация, условие* и *изменение*. Рассмотрим данный цикл на примере:

```
for (int i = 1; i < 7; i++){
    //любые действия
}
```

В *инициализации* мы у нас указана `int i = 1,` т. е. создание счетчика. Значение *i* будет равно 1 перед началом цикла. Тип данных может быть любой и необязательно integer, что зависит от требований к точности вычислений.

Условие – цикл будет выполняться, пока счетчик не достигнет значение «7».

И, наконец, *изменение* – при работе цикла, значение счетчика постоянно будет увеличиваться на 1 (инкремент).

4.1.2 Цикл do

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

```
int x = 14;
do{
    System.out.println(x);
    x--;
}
while (x > 0);
```

В данном случае код цикла сработает 14 раз, пока *x* не окажется равным нулю. Важно отметить, что цикл do гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции while не будет истинно.

4.1.3. Цикл while

While сначала проверяет истинность условия. Если оно истинно, то код выполняется.

```
int x = 14;
while (x > 0){

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

В качестве примера попробуем вывести последовательность чисел от *1* до *100* с помощью цикла *while*:

```
public class Use {
```

```
    public static void main(String[] args) {
        // Ниже объявляем переменную и задаем цикл
        int ch = 1;
        while (ch <= 100){
            System.out.println(ch);
            ch++;
        }
    }
}
```

4.2. Вопросы и задачи к заданию 4

1. Какие действия выполняет оператор *break*?

2. Скажите, что выведет данный код? Ответ обоснуйте.

```
int i = 7;
i = i++;
System.out.println(i);
```

3. Дан одномерный массив {14, 27, 8, 16, 59, 121, 256, 45, 89, 1024}. Напишите программу, которая отсортирует данный массив по убыванию, исключив все те значения, которые больше «98».

4. Компьютер загадывает случайное число от 100 до 1000. У Вас семь попыток, чтобы угадать его. При каждой попытке должно выводиться сообщение «меньше» или «больше». Для считывания данных с клавиатуры используйте класс *Scanner*.

4.3. Пример выполнения задания 4

1. Существуют два оператора break – с меткой и без метки. Без метки – незамедлительное завершение программы, а с меткой – передача управления в конец помеченного блока.

2. Ответ: 7. Дело в том, что здесь инкрементация происходит не сразу, а лишь после возвращения значения. Если бы не было присвоения значения переменной, а сразу стоял инкремент, то ответ был бы равен 8.

3. Текст программы.

```
 public class Test {
    public static void main(String[] args) {
        int a [] = {14, 27, 8, 16, 59, 121, 256, 45,
89, 1024};
        for (int i=0; i<10; i++){
            for (int j = 9; j>i; j--){
                if (a[i]<a[j]){
                    int b = a[i];
                    a[i] = a[j];
                    a[j] = b;
                }
            }

    if (a[i]<98)
        System.out.println(a[i]);
    }

    }
}
```

5. ЗАДАНИЕ 5. МАССИВЫ

5.1. Теоретическая часть

5.1.1. Общие сведения о массивах

Массивы – набор однотипных значений, в то время как переменная – одиночное значение. Сами массивы подразделяются на следующие типы:

– одномерные массивы;

– многомерные массивы;

– нерегулярные массивы.

5.1.2 Одномерные массивы

Одномерные массивы представляют собой «ленту» однотипных значений. Инициализируются массивы практически так же, как и переменные – указывается *тип данных* и *наименование массива*. Единственное, что используется ключевое слово new – оно выделяет память для соответствующих элементов, идущих после ключевого слова. Пример:

```
int [] test = new int [6];
```

В данном примере мы инициализировали массив с шестью элементами. Важно понимать, что в квадратных скобках указывается *количество элементов массива*, а не конкретное значение.

Также мы можем присвоить конкретному элементу свое значение:

```
int [] test = new int [6];
test [0] = 2;
test [1] = 3;
test [2] = 7;
```

```
test [3] = 4;
test [4] = 18;
test [5] = 9;
```

Учтите, что нумерация элементов массива начинается с 0, а не с 1! Поэтому, чтобы обратиться к пятому элементу массива, нам нужно указать test[4].

5.1.3. Многомерные массивы

Фактически, многомерный массив – это таблица, состоящая из *m* строк и *n* столбцов. Инициализируется такой массив следующим образом:

```
int [][] test = new int [6] [6]; //Первый вариант инициализации
int [][] test = {{0,1,2,3,4,5}, {6,7,8,9,10,11}}; //Второй вариант инициализации
```

Оба варианта инициализации абсолютно равнозначны.

Количество квадратных скобок указывает на размерность массива. Это значит, что в примере, указанном выше, перед нами предстает простая двумерная таблица.

5.1.4 Нерегулярные массивы

Возникают ситуации, когда нам необходимо каждому (или определенному) элементу массива присвоить свой массив с определенным количеством элементов. Фактически получается «Массив в массиве». Делается это следующим образом:

```
int [][] test = new int [6] [6];
```

```
test [0] =  new int [2];
test [1] = new int [3];
test [2] = new int [7];
test [3] = new int [4];
test [4] = new int [18];
test [5] = new int [9];
```

В данном примере мы из простого двумерного массива получили внушительную конструкцию из вложенных массивов.

5.2. Вопросы и задачи к заданию 5

1. Какие действия выполняет оператор *new*?

2. Покажите два способа инициализации одномерного массива, состоящего из 52 элементов типа *integer*.

3. Напишите программу, в которой реализуется алгоритм «пузырьковой сортировки», применительно к символьным строкам.

4. Написать программу, которая уменьшает элементы последнего столбца двумерного массива на 5%.

5.3 Пример выполнения задания 3

1. Оператор *new* выделяет память для объекта и выполняет инициализацию объекта.

2.
```
int x [] = new int [52]; //Первый способ
int [] x = new int [52]; //Второй способ
```

3.
```
Class Test {
    public static void main (String [] args) {
        String strs[] = {«test», «one», «is», «this»};
```

```
            int x, y;
            String t;
            int size;
            size = strs.length;

//Отображаем исходный массив
System.out.println("Первоначальный массив: ");
for (int i=0; I < size; i++)
System.out.println(" " + strs[i]);

//Пузырьковая сортировка
for (x = 1; x < size; x++)
for (y = size -1; y >=x; y--) {
    if (strs[y -1].compareTo(strs[y]) > 0) {
        t = strs[y -1];
        strs[b -1] = strs[y];
        strs[y] = t;
}
}
//Отображаем отсортированный массив
System.out.println("Первоначальный массив: ");
for (int i=0; I < size; i++)
System.out.println(" " + strs[i]);
}
}
```

6. ЗАДАНИЕ 6. ОБРАБОТКА ИСКЛЮЧЕНИЙ TRY/ CATCH/ FINALLY

6.1. Теоретический материал

Иногда в процессе выполнения программы, могут возникнуть различного рода ошибки. Они могут «всплыть» как по вине разработчика, так и без его участия. Данные ошибки называются *исключениями*.

В языке Java предусмотрены специальные инструменты для обработки подобных ситуаций. Одним из таких инструментов является конструкция *try/catch/finally*.

При возникновении исключения в блоке *try* управление переходит в блок *catch*, который может обработать данное исключение.

Блок *finally* является независимым. Он выполняется в любом случае – возникло исключение или же нет.

Для примера напишем программу, которая «перехватит» и обработает исключение, возникающее при выход за границы массива:

```
int[] numbers = new int[4];
try{
    numbers[5]=12;
}
catch(ArrayIndexOutOfBoundsException ex){
    System.out.println("Выход за границы массива");
}
finally{
    System.out.println("Выведется на экран в любом случае");
}
```

6.2. Вопросы и задачи к заданию 6

1. Для чего нужен оператор *throw*?

2. Что такое *NumberFormatException*?

3. Напишите программу, которая считывает данные из файла и выводит их на экран. В случае, если файла нет, то программа должна «перехватить» и обработать данное исключение.

4. Написать программу, вычисляющую факториал натурального числа *zh*, который пользователь вводит с клавиатуры. Для считывания данных с клавиатуры используйте класс *Scanner*.

6.3. Пример выполнения задания 6

1. Оператор *throw* позволяет разработчику создавать и обрабатывать собственные исключения.

2. *NumberFormatException* – ошибка преобразования строки в число, исключительная ситуация.

3.
```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
public class tr {

public static void main( String[] args )
    throws FileNotFoundException, IOException {
        File file = new File( "C:\\test.txt" );
        String name = file.getPath();
        System.out.println(name);
```

```
        BufferedReader br = new BufferedReader (
            new InputStreamReader(
                new FileInputStream( file ), "UTF-8"
            )
        );
        String line = null;
        while ((line = br.readLine()) != null) {
           System.out.println( line );
        }
        br.close();
    }
}
```

7. ЗАДАНИЕ 7. ИСПОЛЬЗОВАНИЕ ENUM

7.1. Вопросы и задача к заданию 7

1. Для чего нужен тип *enum*?

2. Для чего нужен оператор *extends*?

3. Напишите программу, которая выводит список фильмов, удовлетворяющих выбранному пользователем жанру.

4. Создать программу, выводящую на экран случайное четырехзначное число и его наименьшую цифру. (Например, число 3415, наименьшая цифра – 1).

7.2. Пример выполнения задания 7

1. *Enum* – это набор логически связанных между собой констант. Другими словами – перечисление. Сперва идет объявление перечисления с помощью типа *enum* и далее, через запятую, идет список констант:

```
Enum Test {
ONE, TWO, THREE, FOUR, FIVE
}
```

2. Определение оператора *Extends* напрямую связано с одним из базовых понятий объектно-ориентированного программирования – с *наследованием* (необходимо вспомнить определение наследования). *Extends* объявляет наследником одного класса другой класс. К примеру, класс Test_2 может наследовать весь функционал класса Test_1.

```
class Test_2 extends Test_1 {
// ТЕЛО КЛАССА
}
```

```
3. class Film{ //Создаем первый класс

        String name;
        Genre filmGenre;
        String producer;
        int year;

        Film(String name, String producer, int year,
Genre genre){

            filmGenre = genre;
            this.name = name;
            this.producer = producer;
            this.year = year;
        }
    }

    enum Genre {
        БОЕВИК,
        ТРИЛЛЕР,
        ФАНТАСТИКА,
        КОМЕДИЯ,
        МЕЛОДРАМА,
        УЖАСЫ
    }

    import java.util.Scanner;

    public class FilmShow { //Создаем второй класс
```

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

            Film f1 = new Film ("Выживший", "Алехандро Гонсалес Иньяритту", 2016, Genre.ТРИЛЛЕР);
      Film f2 = new Film ("Остров проклятых", Мартин Скорсезе", 2009, Genre.ТРИЛЛЕР);
            Film f3 = new Film ("Начало", "Кристофер Нолан", 2010, Genre.ФАНТАСТИКА);

    Scanner in = new Scanner(System.in);
            System.out.println("Введите номер жанра фильма: ");
            int id = in.nextInt();

            Genre genre = Genre.values()[id];
            System.out.println("Выбран жанр: " + genre);

    if(f1.filmGenre==genre){

                System.out.println(“Подходящий фильм: ” + f1.name );
            }
        }
    }
```

СПИСОК ЛИТЕРАТУРЫ

1. Герберт, Шилдт. Java 8: The Complete Reference/ Шилдт Герберт. – М.: Вильямс, 2015.

2. Брюс, Эккель. Философия Java/ Эккель Брюс. – СПб: Питер, 2015.

3. Роберт, Седжвик. Алгоритмы на Java / Седжвик Роберт, Уэйн Кевин. – М.: Вильямс, 2013.

4. Jeff, Friesen. Beginning Java 7/ Friesen Jeff. – Publisher: Apress, 2011.

5. Роберт, Лафоре. Структуры и алгоритмы Java/ Лафоре Роберт. – СПб: Питер, 2013.

6. Paul, Deitel. Java How To Program. Early Objects/ Deitel Paul, Deitel Harvey. – Publisher: Prentice Hall, 2014.

7. Джошуа, Блох. Java. Эффективное программирование/ Блох Джошуа. – Изд-во: Лори, 2002.