5946

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.Ф. УТКИНА

ОСНОВЫ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Часть 1

Методические указания
к лабораторным и практическим занятиям

Рязань 2020

УДК 004.43

Основы промышленной разработки программного обеспечения. Часть 1: методические указания к лабораторным и практическим занятиям / Рязан. гос. радиотехн. ун-т; сост.: Б.В. Костров, А.С. Бастрычкин, Е.А. Трушина. Рязань, 2020, 60 с.

Содержат методические материалы для подготовки к выполнению лабораторных и практических работ.

Предназначены для бакалавров, обучающихся по направлениям: 02.03.03 «Математическое обеспечение и администрирование информационных систем», 09.03.01 «Информатика и вычислительная техника» и 38.03.05 «Бизнес-информатика», а также для специалистов, обучающихся по направлению 27.05.01 «Специальные организационно-технические системы»

Табл. 4. Библиогр.: 3 назв.

Язык программирования Java, объектно-ориентированное программирование, программное обеспечение, промышленная разработка

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра электронных вычислительных машин Рязанского государственного радиотехнического университета (зав. каф. Б.В. Костров)

Основы промышленной разработки
программного обеспечения

Составители: Костров Борис Васильевич
Бастрычкин Александр Сергеевич
Трушина Евгения Александровна

Рязанский государственный радиотехнический университет.
390005, Рязань, ул. Гагарина, 59/1.
Редакционно-издательский центр РГРТУ.

Содержание

Занятие № 1
Структура программ на языке Java

Цель работы: изучение структуры программ на языке Java, приобретение навыков создания простейших приложений.

1. Теоретическая часть

1.1. Структура программы

Основным строительным блоком программы на языке Java являются **инструкции** (statement). Каждая инструкция выполняет некоторое действие, например, вызовы методов, объявление переменных и присвоение им значений. На конец инструкции компилятору указывает точка с запятой (;).

Распространенной конструкцией является блок кода, который содержит набор инструкций. Он заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками. Например:

```
{
    System.out.println("Hello!");
    System.out.println("Welcome to Java!");
}
```

В этом блоке кода две инструкции, представляющие вызов метода **System.out.println**, который выводит на консоль строки "Hello!" и "Welcome to Java!".

1.1.1. Выполнение программы. Метод main

Java является объектно-ориентированным языком, поэтому всю программу можно представить как набор взаимодействующих между собой классов и объектов. Например, программа может быть определена следующим образом:

```
public class Program{
    public static void main (String args[]){
         System.out.println("Hello Java!");
    }
}
```

Основу программы составляет класс **Program**. При определении класса вначале идет модификатор доступа **public**, который указывает, что данный класс будет доступен всем, то есть мы сможем его запустить из командной строки. Далее идет ключевое слово **class**, а затем название класса. После названия класса идет блок кода, в котором расположено содержимое класса.

Входной точкой в программу на языке Java является метод **main**, который определен в классе **Program**. Именно с него начинается выполнение программы. Он обязательно должен присутствовать в программе, и его заголовок строго определен.

При запуске приложения виртуальная машина Java ищет в главном классе программы метод **main** с подобным заголовком, и после его обнаружения запускает его.

Вначале заголовка метода идет модификатор **public**, который указывает, что метод будет доступен извне. Слово **static** указывает, что метод **main** - статический, а слово **void** - что он не возвращает никакого значения. Далее в скобках у нас идут параметры метода - *String args[]* - это массив ***args***, который хранит значения типа **String**, то есть строки. При запуске программы через этот массив мы можем передать в программу различные данные.

После заголовка метода идет его блок, который содержит набор выполняемых инструкций.

1.1.2. Комментарии

Код программы может содержать комментарии, которые позволяют понять смысл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияния на работу приложения и на его размер.

В Java есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша //. А многострочный комментарий заключается между символами /* текст комментария */. Он может размещаться на нескольких строках. Например:

```
   /*
       многострочный комментарий
       Объявление нового класса,
       который содержит код программы
   */
   public class Program{   // начало объявления класса Program
       // определение метода main
       public static void main (String args[]){    //
объявление нового метода
            System.out.println("Hello Java!");     // вывод
строки на консоль
       }   // конец объявления нового метода
   } // конец объявления класса Program
```

1.2. Командная строка

Для запуска программы можно нажать зеленую кнопку в IDE или воспользоваться командной строкой:

1) скомпилировать файл командой
   javac –d bin src\MainClass.java
2) запустить файл на выполнение командой
   java src\MainClass.java

Строковые аргументы будут передаваться через пробел. Например:
java src\MainClass.java Hello! This is Arg1!

1.3. Консольный ввод/вывод в Java

Наиболее простой способ взаимодействия с пользователем представляет консоль: мы можем выводить на консоль некоторую информацию или, наоборот, считывать с консоли некоторые данные. Для взаимодействия с консолью в Java применяется класс **System**, а его функциональность собственно обеспечивает консольный ввод и вывод.

1.3.1. Вывод на консоль

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 курсора консоли на следующую строку. Например:

```
public class Program {

    public static void main(String[] args) {
        System.out.println("Hello world!");
        System.out.println("Bye world...");
    }
}
```

В метод **println** передается любое значение, как правило, строка, которое надо вывести на консоль. И в данном случае мы получим следующий вывод:

```
Hello world!
Bye world...
```

При необходимости можно и не переводить курсор на следующую строку. В этом случае можно использовать метод **System.out.print()**, который аналогичен **println** за тем исключением, что не осуществляет перевода на следующую строку.

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

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

Нередко необходимо подставлять в строку какие-нибудь данные. Например, у нас есть два числа, и мы хотим вывести их значения на экран:

```
public class Program {
    public static void main(String[] args) {
        int x=5;
        int y=6;
        System.out.println("x=" + x + "; y=" + y);
    }
}
```

Консольный вывод программы:

```
x=5; y=6
```

Но в Java есть также функция для форматированного вывода, унаследованная от языка C: **System.out.printf()**. С ее помощью мы можем переписать предыдущий пример:

```
int x=5;
int y=6;
System.out.printf("x=%d; y=%d \n", x, y);
```

В данном случае символы **%d** обозначают спецификатор, вместо которого подставляет один из аргументов. Спецификаторов и соответствующих им аргументов может быть множество. В данном случае у нас только два аргумента, поэтому вместо первого **%d** подставляет значение переменной x, а вместо второго - значение переменной y. Сама буква **d** означает, что данный спецификатор будет использоваться для вывода целочисленных значений.

Кроме спецификатора **%d** мы можем использовать еще ряд спецификаторов для других типов данных:

- **%x**: для вывода шестнадцатеричных чисел
- **%f**: для вывода чисел с плавающей точкой
- **%e**: для вывода чисел в экспоненциальной форме
- **%c**: для вывода одиночного символа
- **%s**: для вывода строковых значений

Например:

```
public class Program {
    public static void main(String[] args) {
        String name = "Tom";
        int age = 30;
        float height = 1.7f;
        System.out.printf("Name: %s  Age: %d  Height: %.2f
\n", name, age, height);
    }
}
```

При выводе чисел с плавающей точкой мы можем указать количество знаков после запятой, для этого используем спецификатор на **%.2f,** где **.2** указывает, что после запятой будет два знака. В итоге мы получим следующий вывод:

```
Name: Tom  Age: 30  Height: 1,70
```

1.3.2. Ввод с консоли

Для получения ввода с консоли в классе **System** определен объект **in**. Однако непосредственно через объект **System.in** не очень удобно работать, поэтому, как правило, используют класс **Scanner**, который, в свою очередь использует **System.in**. Например, напишем маленькую программу, которая осуществляет ввод чисел:

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Input a number: ");
        int num = in.nextInt();
     System.out.printf("Your number: %d \n", num);
        in.close();
    }
}
```

Так как класс **Scanner** находится в пакете *java.util*, то мы вначале его импортируем с помощью инструкции **import java.util.Scanner**.

Для создания самого объекта **Scanner** в его конструктор передается объект **System.in**. После этого мы можем получать вводимые значения. Например, в данном случае вначале выводим приглашение к вводу и затем получаем вводимое число в переменную **num.**

Чтобы получить введенное число, используется метод **in.nextInt();**, который возвращает введенное с клавиатуры целочисленное значение.

Пример работы программы:

```
Input a number: 5
Your number: 5
```

Класс **Scanner** имеет еще 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
- **nextBoolean()**: считывает значение boolean
- **nextByte()**: считывает введенное число byte
- **nextFloat()**: считывает введенное число float
- **nextShort()**: считывает введенное число short

То есть для ввода значений каждого примитивного типа в классе **Scanner** определен свой метод.

Например, создадим программу для ввода информации о человеке:

```
import java.util.Scanner;
public class Program {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Input name: ");
        String name = in.nextLine();
        System.out.print("Input age: ");
        int age = in.nextInt();
        System.out.print("Input height: ");
        float height = in.nextFloat();
        System.out.printf("Name: %s  Age: %d  Height: %.2f
\n", name, age, height);
        in.close();
    }
}
```

Здесь последовательно вводятся данные типов **String**, **int**, **float** и потом все введенные данные вместе выводятся на консоль. Пример работы программы:

```
Input name: Tom
Input age: 34
Input height: 1,7
Name: Tom  Age: 34  Height: 1,70
```

Обратите внимание, что для ввода значения типа **float** (то же самое относится к типу **double**) применяется число "1,7", где разделителем является запятая, а не "1.7", где разделителем является точка.

В данном случае все зависит от текущей языковой локализации системы. В примере, приведенном выше, русскоязычная локализация, соответственно вводить необходимо числа, где разделителем является запятая. То же самое касается многих других локализаций, например, немецкой, французской и т.д., где применяется запятая.

2. Порядок выполнения работы

1. Изучите теоретическую часть лабораторной работы.

2. Выполните задания практической части лабораторной работы по варианту из табл. 1, где № - номер бригады.

Таблица 1 – Варианты заданий

№	Задания		№	Задания		№	Задания	
	1	2		1	2		1	2
1	6	1	6	1	7	11	2	10
2	5	3	7	6	6	12	1	4
3	4	2	8	5	8	13	6	3
4	3	4	9	4	9	14	5	2
5	2	5	10	3	11	15	4	1

3. Практическая часть

3.1. Задание 1

1. Приветствовать любого пользователя при вводе его имени через командную строку.

2. Отобразить в окне консоли аргументы командной строки в обратном порядке.

3. Вывести заданное количество случайных чисел с переходом и без перехода на новую строку.

4. Ввести пароль из командной строки и сравнить его со строкой-образцом.

5. Ввести целые числа как аргументы командной строки, подсчитать их суммы и произведения. Вывести результат на консоль.

6. Вывести фамилию разработчика, дату и время получения задания, а также дату и время сдачи задания.

3.2. Задание 2

Ввести с консоли n целых чисел. На консоль вывести:

1. Четные и нечетные числа.

2. Наибольшее и наименьшее число.

3. Числа, которые делятся на 3 или на 9.

4. Числа, которые делятся на 5 и на 7.

5. Все трехзначные числа, в десятичной записи которых нет одинаковых цифр.

6. Простые числа.

7. Отсортированные числа в порядке возрастания и убывания.

8. Числа в порядке убывания частоты встречаемости чисел.

9. «Счастливые» числа.

10. Числа-палиндромы, значения которых в прямом и обратном порядке совпадают.

11. Элементы, которые равны полусумме соседних элементов.

4. Содержание отчета

1. Краткие теоретические сведения о базовых понятиях языка программирования Java и структуре простейших программ на Java.
2. Код программ.
3. Результаты выполнения программ с различными исходными данными.
4. Выводы по работе.

5. Контрольные вопросы

1. Что имеется в виду, когда говорится: Java-язык программирования и Java-платформа?
2. Расшифровать аббревиатуры JVM, JDK и JRE. Показать, где они физически расположены и что собой представляют.
3. JVM-JDK-JRE. Кто кого включает и как взаимодействуют.
4. Как связаны имя Java-файла и классы, которые в этом файле объявляются?
5. Как скомпилировать и запустить класс, используя командную строку?
6. Что такое *classpath*? Зачем в переменных среды окружения прописывать пути к установленному JDK?
7. Если в *classpath* есть две одинаковые библиотеки (или разные версии одной библиотеки), объект класса из какой библиотеки создастся?
8. Объяснить различия между терминами «объект» и «ссылка на объект».
9. Какие области памяти использует Java для размещения простых типов, объектов, ссылок, констант, методов, пул строк и т.д.
10. Почему метод **main()** объявлен как **public static void**?
11. Возможно ли в сигнатуре метода **main()** поменять местами слова **static** и **void**?
12. Будет ли вызван метод **main()** при запуске приложения, если слова **static** или **public** отсутствуют?
13. Классы какого пакета импортируются в приложение автоматически?

Занятие № 2
Типы данных и операторы

Цель работы: изучение типов данных и операторов в Java-программах, приобретение навыков создания простейших приложений.

1. Теоретическая часть

1.1. Переменные и константы

Для хранения данных в программе предназначены переменные. Переменная представляет именованную область памяти, которая хранит значение определенного типа. Каждая переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная или диапазон допустимых значений.

Переменные объявляются следующим образом: `тип_данных имя_переменной;`

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 словом языка Java.

Кроме того, при объявлении и последующем использовании надо учитывать, что Java - регистрозависимый язык, поэтому следующие объявления **int num;** и **int NUM**; будут представлять две разных переменных.

Объявив переменную, мы можем присвоить ей значение:

```
int x;      // объявление переменной
x = 10;     // присвоение значения
System.out.println(x);  // 10
```

Также можно присвоить значение переменной при ее объявлении. Этот процесс называется **инициализацией:**

```
int x = 10;     // объявление и инициализация переменной
System.out.println(x);  // 10
```

Если мы не присвоим переменной значение до ее использования, то мы можем получить ошибку.

Через запятую можно объявить (напр.: `int x, y;`) или инициализировать (напр.: `int x = 8, y = 15;`) сразу несколько переменных одного типа. Отличительной особенностью переменных

является то, что мы можем в процессе работы программы изменять их значение.

1.1.1. Ключевое слово var

Начиная с Java 10 в язык было добавлено ключевое слово **var**, которое также позволяет определять переменную. Слово **var** ставится вместо типа данных, а сам тип переменной выводится из того значения, которое ей присваивается.

```
var x = 10;
System.out.println(x);  // 10
```

Например, переменной x присваивается число 10, значит, переменная будет представлять тип **int.**

Но если переменная объявляется с помощью **var**, то мы обязательно должны инициализировать ее, иначе мы получим ошибку, как, например, в следующем случае:

```
var x;      // ! Ошибка, переменная не инициализирована
```

1.1.2. Константы

Кроме переменных, в Java для хранения данных можно использовать **константы**. В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется также, как и переменная, только вначале идет ключевое слово **final**:

```
final int LIMIT = 5;
System.out.println(LIMIT);  // 5
// LIMIT=57; // так мы уже не можем написать, так как LIMIT
- константа
```

Как правило, константы имеют имена в верхнем регистре.

Константы позволяют задать такие переменные, которые не должны больше изменяться. Например, если у нас есть переменная для хранения числа pi, то мы можем объявить ее константой, так как ее значение постоянно.

1.2. Типы данных

Одной из основных особенностей Java является то, что данный язык является строго типизированным. А это значит, что каждая переменная и константа представляет определенный тип и данный тип строго определен. Тип данных определяет диапазон значений, которые может хранить переменная или константа.

Итак, рассмотрим систему встроенных базовых типов данных, которая используется для создания переменных в Java. А она представлена следующими типами.

- **boolean**: хранит значение true или false
- **byte**: хранит целое число от -128 до 127 и занимает 1 байт
- **short**: хранит целое число от -32768 до 32767 и занимает 2 байта
- **int**: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта
- **long**: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт
- **double**: хранит число с плавающей точкой от ±4.9*10-324 до ±1.8*10308 и занимает 8 байт
- **float**: хранит число с плавающей точкой от -3.4*1038 до 3.4*1038 и занимает 4 байта
- **char**: хранит одиночный символ в кодировке UTF-16 и занимает 2 байта, поэтому диапазон хранимых значений от 0 до 65535

При этом переменная может принимать только те значения, которые соответствуют ее типу. Если переменная представляет целочисленный тип, то она не может хранить дробные числа. В качестве разделителя целой и дробной части в дробных литералах используется точка.

1.2.1. Целые числа

Все целочисленные литералы, например, числа 10, 4, -5, воспринимаются как значения типа **int**, однако мы можем присваивать целочисленные литералы другим целочисленным типам: **byte**, **long**, **short**. Однако если мы захотим присвоить переменной типа **long** очень большое число, которое выходит за пределы допустимых значений для типа **int**, то мы столкнемся с ошибкой во время компиляции:

```
long num = 2147483649;
```

Здесь число 2147483649 является допустимым для типа **long**, но выходит за предельные значения для типа **int**. И так как все целочисленные значения по умолчанию расцениваются как значения типа **int**, то компилятор укажет нам на ошибку. Чтобы решить проблему, надо добавить к числу суффикс **l** или **L**, который указывает, что число представляет тип **long**:

```
long num = 2147483649L;
```

Как правило, значения для целочисленных переменных задаются в десятичной системе счисления, однако мы можем применять и другие системы счисления. Для задания шестнадцатеричного значения после символов **0x** указывается число в шестнадцатеричном формате. Таким же образом восьмеричное значение указывается после символа **0**, а двоичное значение - после символов **0b**. Например:

```
int num111 = 0x6F; // 16-тиричная система, число 111
int num8 = 010; // 8-ричная система, число 8
int num13 = 0b1101; // 2-ичная система, число 13
```

Также целые числа поддерживают разделение разрядов числа с помощью знака подчеркивания:

```
int x = 234_567__789;
System.out.println(x);  // 234567789
```

1.2.2. Числа с плавающей точкой

При присвоении переменной типа **float** дробного литерала с плавающей точкой, например, 3.1, 4.5 и т.д., Java автоматически рассматривает этот литерал как значение типа **double**. И чтобы указать, что данное значение должно рассматриваться как **float**, нам надо использовать суффикс **f**:

```
float fl = 30.6f;
double db = 30.6;
```

1.2.3. Символы и строки

В качестве значения переменная символьного типа получает одиночный символ, заключенный в одинарные кавычки: `char ch='e';`.

Кроме того, переменной символьного типа также можно присвоить целочисленное значение от 0 до 65535. В этом случае переменная опять же будет хранить символ, а целочисленное значение будет указывать на номер символа в таблице символов Unicode (UTF-16). Например: `char ch=102; // символ 'f'`

Еще одной формой задания символьных переменных является шестнадцатеричная форма: переменная получает значение в шестнадцатеричной форме, которое следует после символов "\u". Например, `char ch='\u0066';` опять же будет хранить символ 'f'.

Символьные переменные не стоит путать со строковыми, 'a' не идентично "a". Строковые переменные представляют объект **String**, который в отличие от **char** или **int** не является примитивным типом в Java.

Кроме собственно символов, которые представляют буквы, цифры, знаки препинания, прочие символы, есть специальные наборы символов, которые называют управляющими последовательностями. Самая популярная последовательность - "\n". Она выполняет перенос на следующую строку. Например: `String text = "Hello \nworld";`

Например, выведем многострочный текст:

```
String text = "Лишь тот, кем бой за жизнь изведан,\n"+
              "Жизнь и свободу заслужил.";
System.out.println(text);
```

С помощью операции + мы можем присоединить к одному тексту другой, причем продолжение текста может располагаться на следующей строке.

Начиная с версии 15 Java поддерживает тестовые блоки (text blocks) - многострочный текст, облеченный в тройные кавычки. Текстовые блоки позволяют упростить написание многострочного текста:

```
String text = """
                Лишь тот, кем бой за жизнь изведан,
                Жизнь и свободу заслужил.
                """;
System.out.println(text);
```

Весь текстовый блок оборачивается в тройные кавычки, при этом не надо использовать соединение строк или последовательность \n для их переноса. Результат выполнения двух последних программ будет одинаковым.

1.3. Операторы

Во время выполнения операций можно задавать приоритет выполнения с помощью скобок (операции в скобках выполняются раньше). Если скобки отсутствуют, выполняются сначала более приоритетные операции. В табл. 2 приведены операторы в порядке убывания приоритетов. Операции, которые расположены на одном уровне в таблице, выполняются согласно ассоциативности выполнения (слева направо или справа налево).

Таблица 2 – Приоритет операций

Оператор	Описание	Ассоциативность
++, —	постинкремент, постдекремент	справа налево
++, —, +, -, ~, !	преинкремент, предекремент, унарный плюс, унарный минус,	справа налево

Оператор	Описание	Ассоциативность
	поразрядное дополнение, булево «не»	
*, /, %	умножение, деление, остаток от деления	слева направо
+, —	сложение, вычитание	слева направо
<<, >>, >>>	сдвиг влево, сдвиг вправо, беззнаковый сдвиг вправо	слева направо
<, >, <=, >=, instanceof	меньше, больше, меньше или равно, больше или равно, сравнить тип	слева направо
==, !=	равно, не равно	слева направо
&	битовое «и»	слева направо
^	исключающее «или»	слева направо
|	битовое «или»	слева направо
&&	логическое «и»	слева направо
||	логическое «или»	слева направо
?:	тернарный оператор	слева направо
=, +=, -=,*=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	операторы присваивания	справа налево

1.4. 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, определим массив чисел:

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

После объявления массива мы можем 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.

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 мы можем обратиться к любому его элементу по индексу, который передается в квадратных скобках после названия переменной массива:

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

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

Так как у нас массив определен только для 4 элементов, то мы не можем обратиться, например, к шестому элементу: `nums[5] = 5;`. Если попытаемся - получим ошибку.

1.4.1. 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, мы можем использовать это свойство:

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

1.4.2. Многомерные массивы

Кроме одномерных массивов также бывают и многомерные. Например:

```
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. А числа в скобках - на количество строк и столбцов. И также, используя индексы, мы можем использовать элементы массива в программе:

```
// установим 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];
```

1.4.3. Зубчатый массив

Многомерные массивы могут быть также представлены как "зубчатые массивы". В вышеприведенном примере двухмерный массив имел 3 строчки и три столбца, поэтому у нас получалась ровная таблица. Но мы можем каждому элементу в двухмерном массиве присвоить отдельный массив с различным количеством элементов:

```
int[][] nums = new int[3][];
nums[0] = new int[2];
nums[1] = new int[3];
nums[2] = new int[5];
```

1.4.4. 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 ее 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 `for(int i:array)`. В частности, в этой версии мы можем 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]);
}
```

1.4.5. 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. Подобным образом можно перебрать и трехмерные массивы и наборы с большим количеством размерностей.

2. Порядок выполнения работы

1. Изучите теоретическую часть лабораторной работы.

2. Выполните задания практической части лабораторной работы по варианту из табл. 3, где № - номер бригады.

Таблица 3 – Варианты заданий

№	Задания			№	Задания			№	Задания		
	1	2	3		1	2	3		1	2	3
1	1	1	1	6	6	7	8	11	2	1	10
2	2	3	4	7	7	8	6	12	3	2	11
3	3	4	2	8	8	6	7	13	4	5	13
4	4	2	3	9	9	9	9	14	5	6	14
5	5	5	5	10	1	3	12	15	6	4	15

3. Практическая часть

3.1. Задание 1

В приведенных ниже заданиях необходимо вывести внизу фамилию разработчика, дату и время получения задания, а также дату и время сдачи задания. Добавить комментарии в программы в виде /** комментарий */. В заданиях на числа объект можно создавать в виде массива символов.

Ввести n чисел с консоли.

1. Найти самое короткое и самое длинное число. Вывести найденные числа и их длину.

2. Упорядочить и вывести числа в порядке возрастания (убывания) значений их длины.

3. Вывести на консоль те числа, длина которых меньше (больше) средней, а также длину.

4. Найти число, в котором число различных цифр минимально. Если таких чисел несколько, найти первое из них.

5. Найти количество чисел, содержащих только четные цифры, а среди них — количество чисел с равным числом четных и нечетных цифр.

6. Найти число, цифры в котором идут в строгом порядке возрастания. Если таких чисел несколько, найти первое из них.

7. Найти число, состоящее только из различных цифр. Если таких чисел несколько, найти первое из них.

8. Среди чисел найти число-палиндром. Если таких чисел больше одного, найти второе.

9. Найти корни квадратного уравнения. Параметры уравнения передавать с командной строкой.

3.2. Задание 2

1. Вывести на экран таблицу умножения.

2. Вывести элементы массива в обратном порядке.

3. Определить принадлежность некоторого значения k интервалам (n, m], [n, m), (n, m), [n, m].

4. Вывести на экран все числа от 1 до 100, которые делятся на 3 без остатка.

5. Сколько значащих нулей в двоичной записи числа 129?

6. В системе счисления с некоторым основанием десятичное число 81 записывается в виде 100. Найти это основание.

7. Написать код программы, которая бы переводила числа из десятичной системы счисления в любую другую.

8. Написать код программы, которая бы переводила числа одной любой системы счисления в любую другую.

9. Ввести число от 1 до 12. Вывести на консоль название месяца, соответствующего данному числу. Осуществить проверку корректности ввода чисел.

3.3. Задание 3

Ввести с консоли n-размерность матрицы a[n][n]. Задать значения элементов матрицы в интервале значений от - n до n с помощью генератора случайных чисел.

1. Упорядочить строки (столбцы) матрицы в порядке возрастания значений элементов k-го столбца (строки).

2. Выполнить циклический сдвиг заданной матрицы на k позиций вправо (влево, вверх, вниз).

3. Найти и вывести наибольшее число возрастающих\убывающих элементов матрицы, идущих подряд.

4. Найти сумму элементов матрицы, расположенных между первым и вторым положительными элементами каждой строки.

5. Вывести числа от 1 до k в виде матрицы N x N слева направо и сверху вниз.

6. Округлить все элементы матрицы до целого числа.

7. Повернуть матрицу на 90, 180 или 270 градусов против часовой стрелки.

8. Вычислить определитель матрицы.

9. Построить матрицу, вычитая из элементов каждой строки матрицы ее среднее арифметическое.

10. Найти максимальный элемент(ы) в матрице и удалить из матрицы все строки и столбцы, его содержащие.

11. Уплотнить матрицу, удаляя из нее строки и столбцы, заполненные нулями.

12. В матрице найти минимальный элемент и переместить его на место заданного элемента путем перестановки строк и столбцов.

13. Преобразовать строки матрицы таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

14. Найти количество всех седловых точек матрицы (матрица A имеет седловую точку Ai, j, если Ai, j является минимальным элементом в i-й строке и максимальным в j-м столбце).

15. Перестроить матрицу, переставляя в ней строки так, чтобы сумма элементов в строках полученной матрицы возрастала.

4. Содержание отчета

1. Краткие теоретические сведения о типах данных и операторах в Java.

2. Код программ.

3. Результаты выполнения программ с различными исходными данными.

4. Выводы по работе.

5. Контрольные вопросы

1. Какие примитивные типы Java существуют, как создать переменные примитивных типов?

2. Объяснить процедуру, по которой переменные примитивных типов передаются в методы как параметры.

3. Каков размер примитивных типов? Как размер примитивных типов зависит от разрядности платформы?

4. Что такое преобразование (приведение) типов и зачем оно необходимо? Какие примитивные типы не приводятся ни к какому другому типу.

5. Объяснить, что такое явное и неявное приведение типов, привести примеры, когда такое преобразование имеет место.

6. Что такое литералы в Java-программе? Дать описание классификации литералов.

7. Как записываются литералы различных видов и типов в Java-программе?

8. Как осуществляется работа с типами при вычислении арифметических выражений в Java?

9. Что такое классы-оболочки, для чего они предназначены? Что значит: объект класса оболочки — константный объект.

10. Объяснить разницу между примитивными и ссылочными типами данных. Пояснить существующие различия, при передаче параметров примитивных и ссылочных типов в методы. Объяснить, как константные объекты ведут себя при передаче в метод.

11. Перечислить известные арифметические, логические и битовые операторы, определить случаи их употребления. Что такое приоритет оператора, как определить, в какой последовательности будут выполняться операции в выражении, если несколько из них имеют одинаковый приоритет.

12. Какие правила выполнения операций с плавающий точкой в Java? Как определить, что результатом вычисления стала бесконечность или «нечисло»?

13. Что такое autoboxing и unboxing? Принцип действия на примерах.

14. Что такое var? Можно ли переменной или методу дать имя **var**? Достоинства и недостатки.

15. Объяснить работу операторов **if, switch, while, do-while, for, for-each**. Написать корректные примеры работы этих операторов.

16. Объяснить работу оператора **instanceof**. Что будет результатом работы оператора, если слева от него будет стоять ссылка, равная **null**?

17. Дать определение массиву. Как осуществляется индексация элементов массива. Как необходимо обращаться к i-ому элементу массива?

18. Привести способы объявления и инициализации одномерных и двумерных массивов примитивных и ссылочных типов. Чем отличаются массивы примитивных и ссылочных типов?

19. Объяснить, что представляет собой двумерный массив в Java, что такое «массив массивов». Как узнать количество строк и количество элементов в каждой строке для «массива массивов»?

20. Объяснить ситуации, когда в коде Java могут возникнуть следующие исключительные ситуации **java.lang.ArrayIndexOutOfBoundsException** и **java.lang.ArrayStoreException**.

Занятие № 3
Классы и методы

Цель работы: изучение принципов ООП, приобретение навыков работы с классами и методами в Java-программах.

1. Теоретическая часть

1.1. Классы и объекты

Java является объектно-ориентированным языком, поэтому такие понятия как "класс" и "объект" играют в нем ключевую роль. Любую программу на Java можно представить как набор взаимодействующих между собой объектов.

Шаблоном или описанием объекта является **класс**, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке - наличие двух рук, двух ног, головы, туловища и т.д. Есть некоторый шаблон - этот шаблон можно назвать классом. Реально же существующий человек (фактически экземпляр данного класса) является объектом этого класса.

Класс определяется с помощью ключевого слова **class**. После названия класса идут фигурные скобки, между которыми помещается тело класса - то есть его поля и методы.

Любой объект может обладать двумя основными характеристиками: состояние - некоторые данные, которые хранит объект, и поведение - действия, которые может совершать объект.

Для хранения состояния объекта в классе применяются поля или переменные класса. Для определения поведения объекта в классе применяются методы.

Например, определим класс **Person**, который представляет человека, и используем его в следующей программе:

```
public class Program{
    public static void main(String[] args) {
        Person tom;
    }
}
```

```
   class Person{
       String name;    // имя
       int age;        // возраст
       void displayInfo(){
           System.out.printf("Name: %s \tAge: %d\n", name,
age);
       }
   }
```

В классе **Person** определены два поля: *name* представляет имя человека, а *age* - его возраст. И также определен метод **displayInfo**, который ничего не возвращает и просто выводит эти данные на консоль.

Как правило, классы определяются в разных файлах. В данном случае для простоты мы определяем два класса в одном файле. Стоит отметить, что в этом случае только один класс может иметь модификатор **public** (в данном случае это класс **Program**), а сам файл кода должен называться по имени этого класса, то есть в данном случае файл должен называться *Program.java*.

Класс представляет новый тип, поэтому мы можем определять переменные, которые представляют данный тип. Так, здесь в методе **main** определена переменная **tom**, которая представляет класс **Person**. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение **null**. По большому счету мы ее пока не можем использовать, поэтому вначале необходимо создать объект класса **Person**.

1.2. Методы

Если переменные и константы хранят некоторые значения, то методы содержат собой набор операторов, которые выполняют определенные действия.

Общее определение методов выглядит следующим образом:

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

Модификаторы и параметры необязательны.

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.*

В фигурные скобки заключено тело метода - все действия, которые он выполняет.

Например, определим и выполним несколько методов:

```
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");
    }
}
```

Здесь определены два дополнительных метода: ***hello*** и ***welcome***, каждый из которых выводит некоторую строку на консоль. Методы определяются внутри класса - в данном случае внутри класса **Program**, в котором определен метод **main**.

Вызов метода осуществляется в форме: `имя_метода(аргументы);`

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

В методе **main** вызывается один раз метод ***hello*** и два раза метод ***welcome***. В этом и заключается одно из преимуществ методов: мы можем вынести некоторые общие действия в отдельный метод и затем вызывать многократно их 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**.

В итоге после компиляции и выполнения программы мы увидим на консоли:

```
Hello
Welcome to Java 10
Welcome to Java 10
```

1.3. Конструкторы

Кроме обычных методов классы могут определять специальные методы, которые называются **конструкторами**.

Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор без параметров.

Выше определенный класс **Person** не имеет никаких конструкторов. Поэтому для него автоматически создается конструктор по умолчанию, который мы можем использовать для создания объекта **Person.** Создадим один объект:

```
public class Program{
    public static void main(String[] args) {
        Person tom = new Person(); // создание объекта
        tom.displayInfo();
        // изменяем имя и возраст
        tom.name = "Tom";
        tom.age = 34;
        tom.displayInfo();
    }
}
class Person{
    String name;    // имя
    int age;        // возраст
    void displayInfo(){
        System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

Для создания объекта **Person** используется выражение **new Person()**. Оператор **new** выделяет память для объекта **Person**. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров.

В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта **Person**. А переменная **tom** получит ссылку на созданный объект.

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

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

В итоге мы увидим на консоли:

```
Name: null                  Age: 0
Name: Tom                   Age: 34
```

Если необходимо, чтобы при создании объекта производилась какая-то логика, например, чтобы поля класса получали какие-то определенные значения, то можно определить в классе свои конструкторы. Например:

```
   public class Program{
       public static void main(String[] args) {
           Person bob = new Person(); // вызов первого
конструктора без параметров
           bob.displayInfo();
           Person tom = new Person("Tom"); // вызов второго
конструктора с одним параметром
           tom.displayInfo();
           Person sam = new Person("Sam", 25); // вызов
третьего конструктора с двумя параметрами
           sam.displayInfo();
       }
   }
   class Person{
       String name;    // имя
       int age;        // возраст
       Person()
       {
           name = "Undefined";
           age = 18;
       }
       Person(String n)
       {
           name = n;
           age = 18;
       }
       Person(String n, int a)
       {
           name = n;
           age = a;
       }
       void displayInfo(){
          System.out.printf("Name: %s \tAge: %d\n", name, age);
       }
   }
```

Теперь в классе определено три конструктора, каждый из которых принимает различное количество параметров и устанавливает значения полей класса.

Консольный вывод программы:

```
Name: Undefined                 Age: 18
Name: Tom                       Age: 18
Name: Sam                       Age: 25
```

1.4. Ключевое слово this

Ключевое слово **this** представляет ссылку на текущий экземпляр класса. Через него мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы. Например:

```
public class Program{
    public static void main(String[] args) {
        Person undef = new Person();
        undef.displayInfo();
        Person tom = new Person("Tom");
        tom.displayInfo();
        Person sam = new Person("Sam", 25);
        sam.displayInfo();
    }
}
class Person{
    String name;    // имя
    int age;        // возраст
    Person()
    {
        this("Undefined", 18);
    }
    Person(String name)
    {
        this(name, 18);
    }
    Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
    void displayInfo(){
       System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

В третьем конструкторе параметры называются так же, как и поля класса. Чтобы разграничить поля и параметры, применяется ключевое слово **this**: `this.name = name;`. Так, в данном случае указываем, что значение параметра **name** присваивается полю **name**.

Кроме того, у нас три конструктора, которые выполняют идентичные действия: устанавливают поля **name** и **age**. Чтобы избежать повторов, с помощью **this** можно вызвать один из конструкторов класса и передать значения для его параметров:

```
Person(String name)
{
    this(name, 18);
}
```

В итоге результат программы будет тот же, что и в предыдущем примере.

1.5. Инициализаторы

Кроме конструктора начальную инициализацию объекта вполне можно было проводить с помощью инициализатора объекта. Инициализатор выполняется до любого конструктора. То есть в инициализатор мы можем поместить код, общий для всех конструкторов:

```
public class Program{
    public static void main(String[] args) {
        Person undef = new Person();
        undef.displayInfo();
        Person tom = new Person("Tom");
        tom.displayInfo();
    }
}
class Person{
    String name;    // имя
    int age;        // возраст
    /*начало блока инициализатора*/
    {
        name = "Undefined";
        age = 18;
    }
    /*конец блока инициализатора*/
    Person(){
    }
    Person(String name){
        this.name = name;
    }
    Person(String name, int age){
        this.name = name;
        this.age = age;
    }
    void displayInfo(){
       System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

Консольный вывод:

```
Name: Undefined                 Age: 18
Name: Tom                       Age: 18
```

1.6. Объекты как параметры методов

Объекты классов, как и данные примитивных типов, могут передаваться в методы. Однако в данном случае есть одна особенность

- при передаче объектов в качестве значения передается копия ссылки на область в памяти, где расположен этот объект. Рассмотрим небольшой пример.

Пусть у нас есть следующий класс **Person**:

```
public class Program{
    public static void main(String[] args) {
        Person kate = new Person("Kate");
        System.out.println(kate.getName());  // Kate
        changeName(kate);
        System.out.println(kate.getName());  // Alice
    }
    static void changeName(Person p){
        p.setName("Alice");
    }
}
class Person{
    private String name;
    Person(String name){
        this.name = name;
    }
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
}
```

Здесь в метод **changeName** передается объект **Person**, у которого изменяется имя. Так как в метод будет передаваться копия ссылки на область памяти, в которой находится объект **Person**, то переменная **kate** и параметр **p** метода **changeName** будут указывать на один и тот же объект в памяти. Поэтому после выполнения метода у объекта **kate**, который передается в метод, будет изменено имя с *"Kate"* на *"Alice"*.

От этого случая следует отличать другой случай:

```
public class Program{
    public static void main(String[] args) {
        Person kate = new Person("Kate");
        System.out.println(kate.getName()); // Kate
        changePerson(kate);
        System.out.println(kate.getName()); // Kate -
изменения не произошло, kate хранит ссылку на старый объект
    }
    static void changePerson(Person p){
        p = new Person("Alice"); // p указывает на новый
объект
        p.setName("Ann");
    }
```

```
    static void changeName(Person p){
        p.setName("Alice");
    }
}
class Person{
    private String name;
    Person(String name){
        this.name = name;
    }
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
}
```

В метод **changePerson** также передается копия ссылки на объект **Person**. Однако в самом методе мы изменяем не отдельные значения объекта, а пересоздаем объект с помощью конструктора и оператора **new**. В результате в памяти будет выделено новое место для нового объекта **Person**, и ссылка на этот объект будет присвоена параметру **p**:

```
static void changePerson(Person p){
    p = new Person("Alice"); // p указывает на новый объект
    p.setName("Ann");        // изменяется новый объект
}
```

То есть после создания нового объекта **Person** параметр **p** и переменная **kate** в методе **main** будут хранить ссылки на разные объекты. Переменная **kate**, которая передавалась в метод, продолжит хранить ссылку на старый объект в памяти. Поэтому ее значение не меняется.

2. Порядок выполнения работы

1. Изучите теоретическую часть лабораторной работы.

2. Выполните задания практической части лабораторной работы по варианту из табл. 4, где № - номер бригады.

Таблица 4 – Варианты заданий

№	Задания			№	Задания			№	Задания		
	1	2	3		1	2	3		1	2	3
1	1	1	1	6	6	7	8	11	11	2	10
2	2	3	4	7	7	8	6	12	12	10	11
3	3	4	2	8	8	6	7	13	1	3	13
4	4	2	3	9	9	9	9	14	2	5	14
5	5	5	5	10	10	1	12	15	3	6	15

3. Практическая часть

3.1. Задание 1

Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы **set***Tun***()**, **get***Tun***()**, **toString()**. Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль. В каждом классе, обладающем информацией, должно быть объявлено несколько конструкторов.

1. **Student**: id, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Факультет, Курс, Группа.

Создать массив объектов. Вывести:

a) список студентов заданного факультета;

b) списки студентов для каждого факультета и курса;

c) список студентов, родившихся после заданного года;

d) список учебной группы.

2. **Customer**: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Номер банковского счета.

Создать массив объектов. Вывести:

a) список покупателей в алфавитном порядке;

b) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

3. **Patient**: id, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз.

Создать массив объектов. Вывести:

a) список пациентов, имеющих данный диагноз;

b) список пациентов, номер медицинской карты которых находится в заданном интервале.

4. **Abiturient**: id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки.

Создать массив объектов. Вывести:

a) список абитуриентов, имеющих неудовлетворительные оценки;

b) список абитуриентов, у которых сумма баллов выше заданной;

c) выбрать заданное число n абитуриентов, имеющих самую высокую сумму баллов (вывести также полный список абитуриентов, имеющих полупроходную сумму).

5. **Book**: id, Название, Автор(ы), Издательство, Год издания, Количество страниц, Цена, Тип переплета.

Создать массив объектов. Вывести:

a) список книг заданного автора;

b) список книг, выпущенных заданным издательством;

c) список книг, выпущенных после заданного года.

6. **House**: id, Номер квартиры, Площадь, Этаж, Количество комнат, Улица, Тип здания, Срок эксплуатации.

Создать массив объектов. Вывести:

a) список квартир, имеющих заданное число комнат;

b) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;

c) список квартир, имеющих площадь, превосходящую заданную.

7. **Phone**: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Дебет, Кредит, Время городских и междугородных разговоров.

Создать массив объектов. Вывести:

a) сведения об абонентах, у которых время внутригородских разговоров превышает заданное;

b) сведения об абонентах, которые пользовались междугородной связью;

c) сведения об абонентах в алфавитном порядке.

8. **Car**: id, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер.

Создать массив объектов. Вывести:

a) список автомобилей заданной марки;

b) список автомобилей заданной модели, которые эксплуатируются больше n лет;

c) список автомобилей заданного года выпуска, цена которых больше указанной.

9. **Product**: id, Наименование, UPC, Производитель, Цена, Срок хранения, Количество.

Создать массив объектов. Вывести:

a) список товаров для заданного наименования;

b) список товаров для заданного наименования, цена которых не превосходит заданную;

c) список товаров, срок хранения которых больше заданного.

10. **Train**: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс).

Создать массив объектов. Вывести:

a) список поездов, следующих до заданного пункта назначения;

b) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

c) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

11. **Bus**: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

Создать массив объектов. Вывести:

a) список автобусов для заданного номера маршрута;

b) список автобусов, которые эксплуатируются больше заданного срока;

c) список автобусов, пробег у которых больше заданного расстояния.

12. **Airline**: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели.

Создать массив объектов. Вывести:

a) список рейсов для заданного пункта назначения;

b) список рейсов для заданного дня недели;

c) список рейсов для заданного дня недели, время вылета для которых больше заданного.

3.2. Задание 2

Реализовать методы сложения, вычитания, умножения и деления объектов (для тех классов, объекты которых могут поддерживать арифметические действия).

1. Определить класс **Дробь (Рациональная Дробь)** в виде пары чисел m и n. Объявить и инициализировать массив из k дробей, ввести/вывести значения для массива дробей. Создать массив/список/множество объектов и передать его в метод, который изменяет каждый элемент массива с четным индексом путем добавления следующего за ним элемента.

2. Определить класс **Комплекс**. Создать массив/список/множество размерности n из комплексных координат. Передать его в метод, который выполнит сложение/умножение его элементов.

3. Определить класс **Квадратное уравнение**. Реализовать методы для поиска корней, экстремумов, а также интервалов убывания/возрастания. Создать массив/список/множество объектов и определить наибольшие и наименьшие по значению корни.

4. Определить класс **Полином** степени n. Объявить массив/список/множество из m полиномов и определить сумму полиномов массива.

5. Определить класс **Интервал** с учетом включения/невключения концов. Создать методы по определению пересечения и объединения интервалов, причем интервалы, не имеющие общих точек, пересекаться/объединятся не могут. Объявить массив/список/множество и n интервалов и определить расстояние между самыми удаленными концами.

6. Определить класс **Точка** на плоскости (в пространстве) и во времени. Задать движение точки в определенном направлении. Создать методы по определению скорости и ускорения точки. Проверить для двух точек возможность пересечения траекторий. Определить расстояние между двумя точками в заданный момент времени.

7. Определить класс **Треугольник** на плоскости. Определить площадь и периметр треугольника. Создать массив/список/множество объектов и подсчитать количество треугольников разного типа (равносторонний, равнобедренный, прямоугольный, произвольный). Определить для каждой группы наибольший и наименьший по площади (периметру) объект.

8. Определить класс **Четырехугольник** на плоскости. Определить площадь и периметр четырехугольника. Создать массив/список/множество объектов и подсчитать количество четырехугольников разного типа (квадрат, прямоугольник, ромб, произвольный). Определить для каждой группы наибольший и наименьший по площади (периметру) объект.

9. Определить класс **Окружность** на плоскости. Определить площадь и периметр. Создать массив/список/множество объектов и определить группы окружностей, центры которых лежат на одной прямой. Определить наибольший и наименьший по площади (периметру) объект.

10. Определить класс **Прямая** на плоскости (пространстве). Определить точки пересечения прямой с осями координат. Определить координаты пересечения двух прямых. Создать массив/список/множество объектов и определить группы параллельных прямых.

3.3. Задание 3

1. Определить класс **Полином** с коэффициентами типа **Рациональная Дробь**. Объявить массив/список/множество из n полиномов и определить сумму полиномов массива.

2. Определить класс **Прямая** на плоскости (в пространстве), параметры которой задаются с помощью **Рациональной Дроби.** Определить точки пересечения прямой с осями координат. Определить координаты пересечения двух прямых. Создать массив/список/множество объектов и определить группы параллельных прямых.

3. Определить класс **Полином** с коэффициентами типа **Комплексное число**. Объявить массив/список/множество из m полиномов и определить сумму полиномов массива.

4. Определить класс **Дробь** в виде пары (m, n) с коэффициентами типа **Комплексное число**. Объявить и инициализировать массив из k дробей, ввести/вывести значения для массива дробей. Создать массив/список/множество объектов и передать его в метод, который изменяет каждый элемент массива с четным индексом путем добавления следующего за ним элемента.

5. Определить класс **Комплекс**, действительная и мнимая часть которой представлены в виде **Рациональной Дроби**. Создать массив/список/множество размерности n из комплексных координат. Передать его в метод, который выполнит сложение/умножение его элементов.

6. Определить класс **Окружность** на плоскости, координаты центра которой задаются с помощью **Рациональной Дроби**. Определить площадь и периметр. Создать массив/список/множество объектов и определить группы окружностей, центры которых лежат на одной прямой. Определить наибольший и наименьший по площади (периметру) объект.

7. Определить класс **Точка** в пространстве, координаты которой задаются с помощью **Рациональной Дроби**. Создать методы по определению расстояния между точками и расстояния до начала координат. Проверить для трех точек возможность нахождения на одной прямой.

8. Определить класс **Точка** в пространстве, координаты которой задаются с помощью Комплексного числа. Создать методы по определению расстояния между точками и расстояния до начала координат.

9. Определить класс **Треугольник на плоскости**, вершины которого имеют тип Точка. Определить площадь и периметр треугольника. Создать массив/список/множество объектов и подсчитать количество треугольников разного типа (равносторонний, равнобедренный, прямоугольный, произвольный). Определить для каждой группы наибольший и наименьший по площади (периметру) объект.

10. Определить класс **Четырехугольник** на плоскости, вершины которого имеют тип **Точка**. Определить площадь и периметр четырехугольника. Создать массив/список/множество объектов и подсчитать количество четырехугольников разного типа (квадрат,

прямоугольник, ромб, произвольный). Определить для каждой группы наибольший и наименьший по площади (периметру) объект.

11. Определить класс **Вектор**. Реализовать методы инкремента, декремента, индексирования. Определить массив из m объектов. Каждую из пар векторов передать в методы, возвращающие их скалярное произведение и длины. Вычислить и вывести углы между векторами.

12. Определить класс **Вектор**. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Объявить массив объектов. Написать метод, который для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными.

13. Определить класс **Булева матрица (BoolMatrix)**. Реализовать методы для логического сложения (дизъюнкции), умножения и инверсии матриц. Реализовать методы для подсчета числа единиц в матрице и упорядочения строк в лексикографическом порядке.

14. Построить класс **Булев вектор (BoolVector)**. Реализовать методы для выполнения поразрядных конъюнкции, дизъюнкции и отрицания векторов, а также подсчета числа единиц и нулей в векторе.

15. Определить класс **Множество символов**. Реализовать методы для определения принадлежности заданного элемента множеству; пересечения, объединения, разности двух множеств. Создать методы сложения, вычитания, умножения (пересечения), индексирования, присваивания. Создать массив объектов и передавать пары объектов в метод другого класса, который строит множество, состоящее из элементов, входящих только в одно из заданных множеств.

4. Содержание отчета

1. Краткие теоретические сведения о классах и методах в Java.
2. Код программ.
3. Результаты выполнения программ с различными исходными данными.
4. Выводы по работе.

5. Контрольные вопросы

1. Дать определение таким понятиям как «класс» и «объект». Привести примеры объявления класса и создания объекта класса. Какие спецификаторы можно использовать при объявлении класса?

2. Как определить, какие поля и методы необходимо определить в классе? Привести пример. Какие спецификаторы можно использовать с полями, а какие с методами (и что они значат)?

3. Что такое конструктор? Как отличить конструктор от любого другого метода? Сколько конструкторов может быть в классе?

4. Что такое конструктор по умолчанию? Может ли в классе совсем не быть конструкторов? Объяснить, какую роль выполняет оператор **this()** в конструкторе?

5. Какова процедура инициализации полей класса и полей экземпляра класса? Когда инициализируются поля класса, а когда — поля экземпляров класса? Какие значения присваиваются полям по умолчанию? Где еще в классе полям могут быть присвоены начальные значения?

6. JavaBeans: основные требования к классам Bean-компонентов, соглашения об именах.

7. В каких областях памяти хранятся значения и объекты, массивы?

8. Дать определение перегрузке методов. Чем удобна перегрузка методов? Указать, какие методы могут перегружаться, и какими методами они могут быть перегружены?

9. Можно ли перегрузить методы в базовом и производном классах? Можно ли **private** метод базового класса перегрузить **public** методом производного?

10. Можно ли перегрузить конструкторы, и можно ли при перегрузке конструкторов менять атрибуты доступа у конструкторов?

11. Свойства конструктора. Способы его вызова.

12. Объяснить, что такое раннее и позднее связывание? Перегрузка — это раннее или позднее связывание?

13. Объяснить правила, которым следует компилятор при разрешении перегрузки; в том числе, если аргументы методов перегружаются примитивными типами, между которыми возможно неявное приведение, или ссылочными типами, состоящими в иерархической связи.

14. Перегрузка. Можно ли менять модификатор доступа метода, если да, то каким образом?

15. Перегрузка. Можно ли менять возвращаемый тип метода, если да, то как? Можно ли менять тип передаваемых параметров?

16. Объяснить, что такое неявная ссылка **this**? В каких методах эта ссылка присутствует, а в каких — нет, и почему?

17. Что такое финальные поля, какие поля можно объявить со спецификатором **final**? Где можно инициализировать финальные поля?

18. Что такое статические поля, статические финальные поля и статические методы. К чему имеют доступ статические методы? Можно ли перегрузить и переопределить статические методы? Наследуются ли статические методы?

19. Что такое логические и статические блоки инициализации? Сколько их может быть в классе, в каком порядке они могут быть размещены и в каком порядке вызываются?

20. Что представляют собой методы с переменным числом параметров, как передаются параметры в такие методы, и что представляет собой такой параметр в методе? Как осуществляется выбор подходящего метода, при использовании перегрузки для методов с переменным числом параметров?

21. Каким образом передаются переменные в методы, по значению или по ссылке?

22. **Mutable** и **Immutable** классы. Привести примеры. Как создать класс, который будет **immutable**?

23. Какие свойства у класса, объявленного как **record**?

24. Что такое **static**? Что будет, если значение атрибута изменить через объект класса? Всегда ли **static** поле содержит одинаковые значения для всех его объектов?

25. **Generics**. Что это такое и для чего применяются. Во что превращается во время компиляции и выполнения? Использование **wildcards**.

26. Что такое **enum**? Какими свойствами обладает? Область применения.

27. Класс **Optional**. Как помогает бороться с проблемой возвращения методом значения **null**?

Занятие № 4
Наследование и полиморфизм

Цель работы: изучение принципов ООП, приобретение навыков использования наследования и полиморфизма в Java-программах.

1. Теоретическая часть

1.1. Наследование

Одним из ключевых аспектов объектно-ориентированного программирования является *наследование*. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Например, имеется следующий класс **Person**, описывающий отдельного человека:

```
class Person {
    private String name;
    public String getName(){ return name; }
    public Person(String name){
        this.name=name;
    }
```

```
    public void display(){
        System.out.println("Name: " + name);
    }
}
```

И, возможно, впоследствии мы захотим добавить еще один класс, который описывает сотрудника предприятия - класс **Employee**. Так как этот класс реализует тот же функционал, что и класс **Person**, поскольку сотрудник - это также и человек, то было бы рационально сделать класс **Employee** производным (наследником, подклассом) от класса **Person**, который, в свою очередь, называется базовым классом, родителем или суперклассом:

```
class Employee extends Person{
}
```

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово **extends**, после которого идет имя базового класса. Для класса **Employee** базовым является **Person**, и поэтому класс **Employee** наследует все те же поля и методы, которые есть в классе **Person.**

Использование классов:

```
public class Program{
    public static void main(String[] args) {
        Person tom = new Person("Tom");
        tom.display();
        Employee sam = new Employee("Sam");
        sam.display();
    }
}
class Person {
    private String name;
    public String getName(){ return name; }
    public Person(String name){
        this.name=name;
    }
    public void display(){
        System.out.println("Name: " + name);
    }
}
class Employee extends Person{
}
```

Производный класс имеет доступ ко всем методам и полям базового класса (даже если базовый класс находится в другом пакете) кроме тех, которые определены с модификатором **private**. При этом производный класс также может добавлять свои поля и методы:

```
   public class Program{
       public static void main(String[] args) {
           Employee sam = new Employee("Sam", "Microsoft");
           sam.display();  // Sam
           sam.work();     // Sam works in Microsoft
       }
   }
   class Person {
       private String name;
       public String getName(){ return name; }
       public Person(String name){
           this.name=name;
       }
       public void display(){
           System.out.println("Name: " + name);
       }
   }
   class Employee extends Person{
       private String company;
       public Employee(String name, String company) {
           super(name);
           this.company=company;
       }
       public void work(){
           System.out.printf("%s works in %s \n", getName(),
company);
       }
   }
```

В данном случае класс **Employee** добавляет поле **company**, которое хранит место работы сотрудника, а также метод **work**.

Если в базовом классе определены конструкторы, то в конструкторе производного классы необходимо вызвать один из конструкторов базового класса с помощью ключевого слова **super**. Например, класс **Person** имеет конструктор, который принимает один параметр. Поэтому в классе **Employee** в конструкторе нужно вызвать констуктор класса **Person**. После слова **super** в скобках идет перечисление передаваемых аргументов. Таким образом, установка имени сотрудника делегируется конструктору базового класса.

При этом вызов конструктора базового класса должен идти в самом начале в конструкторе производного класса.

1.2. Переопределение методов и полиморфизм

Производный класс может определять свои методы, а может переопределять методы, которые унаследованы от базового класса. Например, переопределим в классе **Employee** метод **display**:

```
public class Program{
    public static void main(String[] args) {
        Employee sam = new Employee("Sam", "Microsoft");
        sam.display();  // Sam
                        // Works in Microsoft
    }
}
class Person {
    private String name;
    public String getName(){ return name; }
    public Person(String name){
        this.name=name;
    }
    public void display(){
        System.out.println("Name: " + name);
    }
}
class Employee extends Person{
    private String company;
    public Employee(String name, String company) {
        super(name);
        this.company=company;
    }
    @Override
    public void display(){
        System.out.printf("Name: %s \n", getName());
        System.out.printf("Works in %s \n", company);
    }
}
```

На переопределяемый метод указывает аннотация **@Override**. При переопределении метод должен иметь уровень доступа не меньше, чем в базовом класса. Если в базовом классе метод имеет модификатор **public**, то и в производном классе будет модификатор **public**.

Однако в данном случае мы видим, что часть метода **display** в **Employee** повторяет действия из метода **display** базового класса. Поэтому мы можем сократить класс **Employee**:

```
class Employee extends Person{
    private String company;
    public Employee(String name, String company) {
        super(name);
        this.company=company;
    }
    @Override
    public void display(){
        super.display();
        System.out.printf("Works in %s \n", company);
    }
}
```

С помощью ключевого слова **super** мы также можем обратиться к реализации методов базового класса.

1.3. Запрет наследования

Хотя наследование очень интересный и эффективный механизм, но в некоторых ситуациях его применение может быть нежелательным.

В этом случае можно запретить наследование с помощью ключевого слова **final**. Например:

```
public final class Person {
}
```

Если бы класс **Person** был бы определен таким образом, то следующий код был бы ошибочным и не сработал, так как мы тем самым запретили наследование:

```
class Employee extends Person{ {
}
```

Кроме запрета наследования можно также запретить переопределение отдельных методов. Например, в примере выше переопределен метод **display()**, запретим его переопределение:

```
public class Person {
    //.....................
    public final void display(){
        System.out.println("Имя: " + name);
    }
}
```

В этом случае класс **Employee** не сможет переопределить метод **display**.

1.4. Динамическая диспетчеризация методов

Наследование и возможность переопределения методов открывают нам большие возможности. Прежде всего мы можем передать переменной суперкласса ссылку на объект подкласса:

```
Person sam = new Employee("Sam", "Oracle");
```

Так как **Employee** наследуется от **Person**, то объект **Employee** является в то же время и объектом **Person**. Грубо говоря, любой работник предприятия одновременно является человеком.

Однако несмотря на то, что переменная представляет объект **Person**, виртуальная машина видит, что в реальности она указывает на объект **Employee**. Поэтому при вызове методов у этого объекта будет вызываться та версия метода, которая определена в классе **Employee**, а не в **Person**. Например:

```
public class Program{
    public static void main(String[] args) {
        Person tom = new Person("Tom");
        tom.display();
        Person sam = new Employee("Sam", "Oracle");
        sam.display();
    }
}
class Person {
    private String name;
    public String getName() { return name; }
    public Person(String name){
        this.name=name;
    }
    public void display(){
        System.out.printf("Person %s \n", name);
    }
}
class Employee extends Person{
    private String company;
    public Employee(String name, String company) {
        super(name);
        this.company = company;
    }
    @Override
    public void display(){
        System.out.printf("Employee %s works in %s \n",
super.getName(), company);
    }
}
```

Консольный вывод данной программы:

```
Person Tom
Employee Sam works in Oracle
```

При вызове переопределенного метода виртуальная машина динамически находит и вызывает именно ту версию метода, которая определена в подклассе.

Данный процесс еще называется **dynamic method lookup** или динамический поиск метода или динамическая диспетчеризация методов.

2. Порядок выполнения работы

1. Изучите теоретическую часть лабораторной работы.

2. Выполните задания практической части лабораторной работы по варианту.

3. Практическая часть

3.1. Задание 1

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Наследование применять только в тех заданиях, в которых это логически обосновано. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы **equals()**, **hashCode()**, **toString()**.

1. Создать объект класса **Автомобиль**, используя классы **Колесо, Двигатель**. Методы: ехать, заправляться, менять колесо, вывести на консоль марку автомобиля.
2. Создать объект класса **Самолет**, используя классы **Крыло, Шасси, Двигатель**. Методы: летать, задавать маршрут, вывести на консоль маршрут.
3. Создать объект класса **Государство**, используя классы **Область, Район, Город**. Методы: вывести на консоль столицу, количество областей, площадь, областные центры.
4. Создать объект класса **Планета**, используя классы **Материк, Океан, Остров**. Методы: вывести на консоль название материка, планеты, количество материков.
5. Создать объект класса **Звездная система**, используя классы **Планета, Звезда, Луна**. Методы: вывести на консоль количество планет в звездной системе, название звезды, добавление планеты в систему.
6. Создать объект класса **Щенок**, используя классы **Животное, Собака**. Методы: вывести на консоль имя, подать голос, прыгать, бегать, кусать.
7. Создать объект класса **Текстовый файл**, используя классы **Файл, Директория**. Методы: создать, переименовать, вывести на консоль содержимое, дополнить, удалить.
8. Создать объект класса **Дом**, используя классы **Окно, Дверь**. Методы: закрыть на ключ, вывести на консоль количество окон, дверей.
9. Создать объект класса **Дерево**, используя классы **Лист, Ветка**. Методы: зацвести, опасть листьям, покрыться инеем, пожелтеть листьям.
10. Создать объект класса **Пианино**, используя классы **Клавиша, Педаль**. Методы: настроить, играть на пианино, нажимать клавишу.
11. Создать объект класса **Фотоальбом**, используя классы **Фотография, Страница**. Методы: задать название фотографии, дополнить фотоальбом фотографией, вывести на консоль количество фотографий.

12. Создать объект класса **Год**, используя классы **Месяц, День**. Методы: задать дату, вывести на консоль день недели по заданной дате, рассчитать количество дней, месяцев в заданном временном промежутке.

13. Создать объект класса **Сутки**, используя классы **Час, Минута**. Методы: вывести на консоль текущее время, рассчитать время суток (утро, день, вечер, ночь).

14. Создать объект класса **Птица**, используя классы **Крылья, Клюв**. Методы: летать, садиться, питаться, атаковать.

15. Создать объект класса **Хищник**, используя классы **Когти, Зубы**. Методы: рычать, бежать, спать, добывать пищу.

3.2. Задание 2

Создать консольное приложение, удовлетворяющее следующим требованиям:

- Использовать возможности ООП: классы, наследование, полиморфизм, инкапсуляция.
- Каждый класс должен иметь отражающее смысл название и информативный состав.
- Наследование должно применяться только тогда, когда это имеет смысл.
- При кодировании должны быть использованы соглашения об оформлении кода java code convention.
- Классы должны быть грамотно разложены по пакетам.
- Консольное меню должно быть минимальным.
- Для хранения параметров инициализации можно использовать файлы.

1. **Цветочница**. Определить иерархию цветов. Создать несколько объектов-цветов. Собрать букет (используя аксессуары) с определением его стоимости. Провести сортировку цветов в букете на основе уровня свежести. Найти цветок в букете, соответствующий заданному диапазону длин стеблей.

2. **Шеф-повар**. Определить иерархию овощей. Сделать салат. Подсчитать калорийность. Провести сортировку овощей для салата на основе одного из параметров. Найти овощи в салате, соответствующие заданному диапазону калорийности.

3. **Звукозапись**. Определить иерархию музыкальных композиций. Записать на диск сборку. Подсчитать продолжительность. Провести перестановку композиций диска на основе принадлежности к стилю. Найти композицию, соответствующую заданному диапазону длины треков.

4. **Камни**. Определить иерархию драгоценных и полудрагоценных камней. Отобрать камни для ожерелья. Подсчитать общий вес (в каратах) и стоимость. Провести сортировку камней ожерелья на основе ценности. Найти камни в ожерелье, соответствующие заданному диапазону параметров прозрачности.

5. **Мотоциклист**. Определить иерархию амуниции. Экипировать мотоциклиста. Подсчитать стоимость. Провести сортировку амуниции на основе веса. Найти элементы амуниции, соответствующие заданному диапазону параметров цены.

6. **Транспорт**. Определить иерархию подвижного состава железнодорожного транспорта. Создать пассажирский поезд. Подсчитать общую численность пассажиров и багажа. Провести сортировку вагонов поезда на основе уровня комфортности. Найти в поезде вагоны, соответствующие заданному диапазону параметров числа пассажиров.

7. **Авиакомпания**. Определить иерархию самолетов. Создать авиакомпанию. Посчитать общую вместимость и грузоподъемность. Провести сортировку самолетов компании по дальности полета. Найти самолет в компании, соответствующий заданному диапазону параметров потребления горючего.

8. **Таксопарк**. Определить иерархию легковых автомобилей. Создать таксопарк. Подсчитать стоимость автопарка. Провести сортировку автомобилей парка по расходу топлива. Найти автомобиль в компании, соответствующий заданному диапазону параметров скорости.

9. **Страхование**. Определить иерархию страховых обязательств. Собрать из обязательств деривати. Подсчитать стоимость. Провести сортировку обязательств в деривативе на основе уменьшения степени риска. Найти обязательство в деривативе, соответствующее заданному диапазону параметров.

10. **Мобильная связь**. Определить иерархию тарифов мобильной компании. Создать список тарифов компании. Подсчитать общую численность клиентов. Провести сортировку тарифов на основе размера абонентской платы. Найти тариф в компании, соответствующий заданному диапазону параметров.

11. **Фургон кофе**. Загрузить фургон определенного объема грузом на определенную сумму из различных сортов кофе, находящихся к тому же в разных физических состояниях (зерно, молотый, растворимый в банках и пакетиках). Учитывать объем кофе вместе с упаковкой. Провести сортировку товаров на основе соотношения цены

и веса. Найти в фургоне товар, соответствующий заданному диапазону параметров качества.

12. **Налоги**. Определить множество и сумму налоговых выплат физического лица за год с учетом доходов с основного и дополнительного мест работы, авторских вознаграждений, продажи имущества, получения в подарок денежных сумм и имущества, переводов из-за границы, льгот на детей и материальной помощи. Провести сортировку налогов по сумме.

13. **Счета**. Клиент может иметь несколько счетов в банке. Учитывать возможность блокировки/разблокировки счета. Реализовать поиск и сортировку счетов. Вычисление общей суммы по счетам. Вычисление суммы по всем счетам, имеющим положительный и отрицательный балансы отдельно.

14. **Туристические путевки**. Сформировать набор предложений клиенту по выбору туристической путевки различного типа (отдых, экскурсии, лечение, шопинг, круиз и т.д.) для оптимального выбора. Учитывать возможность выбора транспорта, питания и числа дней. Реализовать выбор и сортировку путевок.

15. **Кредиты**. Сформировать набор предложений клиенту по целевым кредитам различных банков для оптимального выбора. Учитывать возможность досрочного погашения кредита и/или увеличения кредитной линии. Реализовать выбор и поиск кредита.

4. Содержание отчета

1. Краткие теоретические сведения о наследовании и полиморфизме в Java.
2. Код программ.
3. Результаты выполнения программ с различными исходными данными.
4. Выводы по работе.

5. Контрольные вопросы

1. Принципы ООП.
2. Правила переопределения метода **boolean equals(Object o)**.
3. Зачем переопределять методы **hashCode()** и **equals()** одновременно?
4. Написать метод **equals()** для класса, содержащего одно поле типа **String**.
5. Правила переопределения метода **int hashCode()**. Можно ли в качестве результата возвращать константу?
6. Правила переопределения метода **clone()**.

7. Чем отличаются **finally** и *finalize*? Для чего используется ключевое слово **final**?

8. JavaBeans: основные требования к классам Bean-компонентов, соглашения об именах.

9. Как работает *Garbage Collector*. Какие самые распространенные алгоритмы? Можно ли самому указать сборщику мусора, какой объект удалить из памяти.

10. В каких областях памяти хранятся значения и объекты, массивы?

11. Чем является класс **Object**? Перечислить известные методы класса **Object**, указать их назначение.

12. Что такое хэш-значение? Объяснить, почему два разных объекта могут сгенерировать одинаковые хэш-коды?

13. Для чего используется наследование классов в java-программе? Привести пример наследования. Поля и методы, помеченные модификатором доступа **private**, наследуются?

Занятие № 5
Интерфейсы

Цель работы: изучение методов реализации интерфейсов, приобретение навыков использования интерфейсов в Java-программах.

1. Теоретическая часть

1.1. Интерфейсы

Механизм наследования очень удобен, но он имеет свои ограничения. В частности, мы можем наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование.

В языке Java подобную проблему частично позволяют решить **интерфейсы**.

Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. Один класс может применить множество интерфейсов.

Интерфейс может определять константы и методы, которые могут иметь, а могут и не иметь реализации. Методы без реализации похожи на абстрактные методы абстрактных классов.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ **public**, так как цель интерфейса - определение функционала для реализации его классом.

Чтобы определить интерфейс, используется ключевое слово **interface**.

Чтобы класс применил интерфейс, надо использовать ключевое слово **implements**. Например:

```
public class Program{
    public static void main(String[] args) {
        Book b1 = new Book("Java. Complete Referense.", "H.
Shildt");
        b1.print();
    }
}
interface Printable{
    void print();
}
class Book implements Printable{
    String name;
    String author;
    Book(String name, String author){
        this.name = name;
        this.author = author;
    }
    public void print() {
        System.out.printf("%s (%s) \n", name, author);
    }
}
```

В данном случае класс **Book** реализует интерфейс **Printable**.

Если класс применяет интерфейс, то он должен реализовать все методы интерфейса, как в случае выше реализован метод **print**. Потом в методе **main** мы можем создать объект класса **Book** и вызвать его метод **print**.

Если класс не реализует какие-то методы интерфейса, то он должен быть определен как абстрактный, а его неабстрактные классы-наследники затем должны будут реализовать эти методы.

Одним из преимуществ использования интерфейсов является то, что они позволяют добавить в приложение гибкости.

В дополнение к классу **Book** определим еще один класс, который будет реализовывать интерфейс **Printable**, например, класс **Journal**:

```
class Journal implements Printable {
    private String name;
    String getName(){
        return name;
    }
    Journal(String name){
        this.name = name;
    }
    public void print() {
```

```
        System.out.println(name);
    }
}
```

Класс **Book** и класс **Journal** связаны тем, что они реализуют интерфейс **Printable**.

Теперь мы динамически в программе можем создавать объекты **Printable** как экземпляры обоих классов:

```
public class Program{
    public static void main(String[] args) {
        Printable printable = new Book("Java. Complete
Reference", "H. Shildt");
        printable.print();      //  Java. Complete Reference
(H. Shildt)
        printable = new Journal("Foreign Policy");
        printable.print();      // Foreign Policy
    }
}
interface Printable{
    void print();
}
class Book implements Printable{
    String name;
    String author;
    Book(String name, String author){
        this.name = name;
        this.author = author;
    }
    public void print() {
        System.out.printf("%s (%s) \n", name, author);
    }
}
class Journal implements Printable {
    private String name;
    String getName(){
        return name;
    }
    Journal(String name){
        this.name = name;
    }
    public void print() {
        System.out.println(name);
    }
}
```

1.2. Методы в интерфейсах

1.2.1. Методы по умолчанию

До JDK 8 при реализации интерфейса мы должны были обязательно реализовать все его методы в классе, а сам интерфейс мог содержать только определения методов без конкретной реализации.

В JDK 8 была добавлена такая функциональность как методы по умолчанию. Теперь интерфейсы кроме определения методов могут иметь их реализацию по умолчанию, которая используется, если класс, реализующий данный интерфейс, не реализует метод.

Например, создадим метод по умолчанию в интерфейсе **Printable**:

```
interface Printable {
    default void print(){
        System.out.println("Undefined printable");
    }
}
```

Метод по умолчанию - это обычный метод без модификаторов, который помечается ключевым словом **default**.

Затем в классе **Journal** нам необязательно этот метод реализовывать, хотя мы можем его и переопределить:

```
class Journal implements Printable {
    private String name;
    String getName(){
        return name;
    }
    Journal(String name){
        this.name = name;
    }
}
```

1.2.2. Статические методы

Начиная с JDK 8 в интерфейсах доступны статические методы - они аналогичны методам класса:

```
interface Printable {
    void print();
    static void read(){
        System.out.println("Read printable");
    }
}
```

Чтобы обратиться к статическому методу интерфейса, так же, как и в случае с классами, пишут название интерфейса и метод:

```
public static void main(String[] args) {
    Printable.read();
}
```

1.2.3. Приватные методы

По умолчанию все методы в интерфейсе имеют модификатор **public**. Начиная с Java 9, мы также можем определять в интерфейсе

методы с модификатором **private**. Они могут быть статическими и нестатическими, но они не могут иметь реализации по умолчанию.

Подобные методы могут использоваться только внутри того интерфейса, в котором они определены.

Например, нам надо выполнять в интерфейсе некоторые повторяющиеся действия, и в этом случае такие действия можно выделить в приватные методы:

```
public class Program{
    public static void main(String[] args) {
        Calculatable c = new Calculation();
        System.out.println(c.sum(1, 2));
        System.out.println(c.sum(1, 2, 4));
    }
}
class Calculation implements Calculatable{
}
interface Calculatable{
    default int sum(int a, int b){
        return sumAll(a, b);
    }
    default int sum(int a, int b, int c){
        return sumAll(a, b, c);
    }
    private int sumAll(int... values){
         int result = 0;
         for(int n : values){
             result += n;
         }
         return result;
    }
}
```

1.3. Константы в интерфейсах

Кроме методов в интерфейсах могут быть определены статические константы, которые по умолчанию имеют модификатор доступа **public static final**, и поэтому их значение доступно из любого места программы. Например:

```
public class Program{
    public static void main(String[] args) {
        WaterPipe pipe = new WaterPipe();
        pipe.printState(1);
    }
}
class WaterPipe implements Stateable{
    public void printState(int n){
        if(n==OPEN)
            System.out.println("Water is opened");
        else if(n==CLOSED)
```

```
            System.out.println("Water is closed");
        else
            System.out.println("State is invalid");
    }
}
interface Stateable{
    int OPEN = 1;
    int CLOSED = 0;
    void printState(int n);
}
```

1.4. Интерфейсы в преобразованиях типов

Все сказанное в отношении преобразования типов характерно и для интерфейсов. Например, так как класс **Journal** реализует интерфейс **Printable**, то переменная типа **Printable** может хранить ссылку на объект типа **Journal**:

```
Printable p =new Journal("Foreign Affairs");
p.print();
// Интерфейс не имеет метода getName, необходимо явное
приведение
String name = ((Journal)p).getName();
System.out.println(name);
```

И если мы хотим обратиться к методам класса **Journal**, которые определены не в интерфейсе **Printable**, а в самом классе **Journal**, то нам надо явным образом выполнить преобразование типов: `((Journal)p).getName();`

1.5. Множественная реализация интерфейсов

Если нам надо применить в классе несколько интерфейсов, то они все перечисляются через запятую после слова **implements**:

```
interface Printable {
    // методы интерфейса
}
interface Searchable {
    // методы интерфейса
}
class Book implements Printable, Searchable{
    // реализация класса
}
```

1.6. Наследование интерфейсов

Интерфейсы, как и классы, могут наследоваться:

```
interface BookPrintable extends Printable{
    void paint();
}
```

При применении этого интерфейса класс **Book** должен будет реализовать как методы интерфейса **BookPrintable**, так и методы базового интерфейса **Printable**.

1.7. Вложенные интерфейсы

Как и классы, интерфейсы могут быть ***вложенными***, то есть могут быть определены в классах или других интерфейсах.

Например:

```
class Printer{
    interface Printable {
        void print();
    }
}
```

При применении такого интерфейса нам надо указывать его полное имя вместе с именем класса:

```
public class Journal implements Printer.Printable {
    String name;
    Journal(String name){
        this.name = name;
    }
    public void print() {
        System.out.println(name);
    }
}
```

Использование интерфейса будет аналогично предыдущим случаям:

```
Printer.Printable p =new Journal("Foreign Affairs");
p.print();
```

1.8. Интерфейсы как параметры и результаты методов

Как и в случае с классами, интерфейсы могут использоваться в качестве типа параметров метода или в качестве возвращаемого типа:

```
public class Program{
    public static void main(String[] args) {
        Printable printable = createPrintable("Foreign
Affairs",false);
        printable.print();
        read(new Book("Java for impatients", "Cay Horstmann"));
        read(new Journal("Java Dayly News"));
    }
    static void read(Printable p){
        p.print();
    }
    static Printable createPrintable(String name, boolean
option){
```

```
        if(option)
            return new Book(name, "Undefined");
        else
            return new Journal(name);
    }
}
interface Printable{
    void print();
}
class Book implements Printable{
    String name;
    String author;
    Book(String name, String author){
        this.name = name;
        this.author = author;
    }
    public void print() {
        System.out.printf("%s (%s) \n", name, author);
    }
}
class Journal implements Printable {
    private String name;
    String getName(){
        return name;
    }
    Journal(String name){
        this.name = name;
    }
    public void print() {
        System.out.println(name);
    }
}
```

Метод **read()** в качестве параметра принимает объект интерфейса **Printable**, поэтому в этот метод мы можем передать как объект **Book**, так и объект **Journal**.

Метод **createPrintable()** возвращает объект **Printable**, поэтому также мы можем возвратить как объект **Book**, так и **Journal**.

Консольный вывод:

```
Foreign Affairs
Java for impatients (Cay Horstmann)
Java Dayly News
```

1.9. Интерфейсы в механизме обратного вызова

Одним из распространенных способов использования интерфейсов в Java является создание обратного вызова, когда одни действия вызываются другими действиями.

Стандартный пример - нажатие на кнопку. Например, нажатие на значок принтера запускает печать документа на принтере и т.д.

Рассмотрим следующий пример:

```
public class EventsApp {
    public static void main(String[] args) {
        Button button = new Button(new ButtonClickHandler());
        button.click();
    }
}
class ButtonClickHandler implements EventHandler{
    public void execute(){
        System.out.println("Кнопка нажата!");
    }
}
interface EventHandler{
    void execute();
}
class Button{
    EventHandler handler;
    Button(EventHandler action){
        this.handler=action;
    }
    public void click(){
        handler.execute();
    }
}
```

Определен класс **Button**, который в конструкторе принимает объект интерфейса **EventHandler** и в методе **click** (*имитация нажатия*) вызывает метод **execute** объекта.

Далее определяется реализация **EventHandler** в виде класса **ButtonClickHandler**. В основной программе объект этого класса передается в конструктор **Button**.

Таким образом, через конструктор мы устанавливаем обработчик нажатия кнопки. При каждом вызове метода **button.click()** будет вызываться этот обработчик.

В итоге программа выведет на консоль следующий результат:

```
Кнопка нажата!
```

В данном качестве интерфейсы особенно широко используются в различных графических API - AWT, Swing, JavaFX, где обработка событий объектов - элементов графического интерфейса особенно актуальна.

2. Порядок выполнения работы

1. Изучите теоретическую часть лабораторной работы.

2. Выполните задания практической части лабораторной работы по варианту.

3. Практическая часть

3.1. Задание 1

Разработать проект управления процессами на основе создания и реализации интерфейсов для следующих предметных областей:

1. **Абонент мобильного оператора**. Возможности: оформить договор; открыть счет и номер; редактировать учетную запись абонента; получить всю актуальную информацию по номеру абонента; проверить состояние баланса и остаток трафика; просмотреть детализацию и информацию о платежах; сменить тарифный план, оператора; пополнить счет; закрыть счет и номер. Добавить специализированные методы для Учетной записи интернет и корпоративного абонента.

2. **Авиарейс**. Возможности: получить или изменить различную информацию о рейсе: номер рейса, пункт назначения, марка судна, статус судна (загрузка, разгрузка, заправка, в пути, в ремонте, готов к вылету, требуется ремонт); узнать среднее время рейса; узнать снаряженную массу; узнать количество топлива; отправить в пункт назначения; отремонтировать; заправить; загрузить; разгрузить; узнать хрупкость/ценность груза. Добавить дополнительные возможности для грузового и пассажирского рейса.

3. **Средство передвижения**. Возможности: получить или изменить различную информацию о средстве передвижения: регистрационный номер, марка, модель, VIN-номер, владелец, тип движущей силы; заправить\отремонтировать\обслужить; пройти техосмотр. Добавить дополнительные возможности для автомобиля, велосипеда, самоката, мотоцикла, квадроцикла.

4. **Лекарственный препарат**. Возможности: добавить действующее вещество; рассчитать дозировку; провести исследование вещества; изменить статус вещества (запрещенное, по рецепту, разрешенное); получить и изменить информации о действующем веществе. Добавить дополнительные возможности для Лекарства, Мази, Таблетки и Раствора.

5. **Дом**. Возможности: построить дом; рассчитать цену за квадратный метр; узнать сколько комнат; увеличить площадь; сдавать дом в аренду; сделать ремонт (в какой-либо комнате). Добавить специализированные методы для Дома, Офисного здания, Торгового центра.

6. **Банковский вклад**. Возможности: изменить продолжительность вклада (бессрочный/долгосрочный/краткосрочный); узнать, какой вклад (отзывной или безотзывной); закрыть один вклад и открыть новый с такими же условиями, но в другой валюте; рассчитать начисление процентов по вкладу.

7. **Компьютер**. Возможности: создать новую модель компьютера; установить цену; добавить объем оперативной памяти; изменить разрешение экрана; изменить процессор компьютера; добавить новые комплектующие в базовую комплектацию. Добавить специализированные методы для Computer, SmartPhone, Pad.

8. **Город**. Возможности: вывести основную информацию о городе (страна, год основания, популяция, площадь, расположение, бюджет, действующий мэр, язык общения); выбрать нового мэра города; увеличить/уменьшить популяцию населения; редактировать бюджет; изменить статус города (закрытие на карантин при определенном проценте зараженных); ввести\изменить закон. Добавить специализированные методы для Города, Деревни, Хутора.

9. **Товар (Product)**. Возможности: добавить товар; получить данные о товаре: id, наименование, UPC, производитель, цена, срок хранения, количество; изменить данные о товаре; переместить товар на склад, переместить товар в торговый зал; оплатить товар; списать товар. Добавить специализированные методы для масла, молока, творога.

10. **Пациент (Patient)**. Возможности: регистрировать пациента; оформить договор обслуживания; получить персональные данные: id, фамилия, имя, отчество, адрес, телефон, номер медицинской карты; изменить персональные данные; записаться на прием к врачу; записать диагноз/назначенное лечение; сдать лабораторные исследования; получить историю болезни; получить диагноз/назначение лечение; оплатить услуги; отказаться от обслуживания; возобновить обслуживание. Добавить специализированные методы для Пациента инфекционного отделения, Пациента стационара, Пациента с COVID-19.

11. **Покупатель (Customer)**. Возможности: оформить договор; получить \изменить персональные данные; добавить покупки; получить оплату; добавить скидки; расторгнуть договор; возобновить договор. Добавить специализированные методы для Оптового покупателя, VIP-покупателя.

12. **Планета**. Возможности: добавить планету; сообщить об открытии; добавить/изменить характеристики; рассчитать период вращения; построить модель планеты; изменить модель планеты; заказать дополнительные исследования; купить индивидуальное название; добавить снимки; напечатать отчет о планете; выбрать планеты по параметру. Добавить специализированные методы для Газовых гигантов, Ледяных гигантов, Карликовых планет.

13. **Погода**. Возможности: ввести/получить данные мониторинга метеорологических условий; запросить данные; анализировать карты погоды; анализировать аэрологическую информацию; анализировать

климатические данные; рассчитать прогнозные показатели; уточнить прогнозные показатели; обновить информацию; ввести фактические показатели; отправить на анализ; вывести прогноз погоды; оценить прогноз. Добавить специализированные методы для Срединный регион, Нижний регион, Верхний регион.

14. **Расписание занятий**. Возможности: создать/удалить расписание; создать/ изменить/удалить предмет/преподавателя; отобразить расписание для конкретного преподавателя/конкретной группы; отобразить список всех предметов/преподавателей; изменить аудиторию; отменить занятие; добавить дополнительное занятие; добавить выходной; посчитать количество занятий у преподавателя/группы. Добавить специализированные методы для Расписания занятий в школе, Расписания занятий в автошколе.

15. **Абитуриент РГРТУ**. Возможности: получить аттестат; заполнить заявление; зарегистрироваться\пройти вступительные испытания; подать документы в приемную комиссию. Добавить дополнительные возможности для Абитуриента ИМиА РГРТУ (магистратура), Абитуриента ИМиА РГРТУ (аспирантура).

4. Содержание отчета

1. Краткие теоретические сведения об интерфейсах в Java.
2. Код программ.
3. Результаты выполнения программ с различными исходными данными.
4. Выводы по работе.

5. Контрольные вопросы

1. Что такое интерфейс? Как определить и реализовать интерфейс в java-программе?
2. Можно ли описывать в интерфейсе конструкторы и создавать объекты?
3. Можно ли создавать интерфейсные ссылки и если да, то на какие объекты они могут ссылаться?
4. Какие идентификаторы по умолчанию имеют поля интерфейса?
5. Какие идентификаторы по умолчанию имеют методы интерфейса?
6. Чем отличается абстрактный класс от интерфейса?
7. Когда применять интерфейс логичнее, а когда абстрактный класс?
8. Бывают ли интерфейсы без методов? Для чего?
9. Могут ли классы внутри классов реализовывать интерфейсы?
10. Можно ли создать анонимный класс на основе реализации интерфейса?
11. Как объявить статический метод в интерфейсе (2 способа)?

Библиографический список

1. Java. Промышленное программирование : практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с.

2. Java. Методы программирования : уч.-мет. пособие / И.Н. Блинов, В.С. Романчик. — Минск : издательство «Четыре четверти», 2013. — 896 с.

3. Java from EPAM : учеб.-метод. пособие / И.Н. Блинов, В.С. Романчик. — Минск : Четыре четверти, 2020. — 560 с.