МИНОБРНАУКИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
МИРЭА

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ для студентов, обучающихся по направлению подготовки Программная инженерия 09.03.04

МОСКВА 2018

Составитель Н. В. Зорина, Л. Б. Зорин, В.Л. Хлебникова

Редактор О.В. Соболев

Методические указания содержат задания по лабораторным работам, а также примеры программ на языке Java. Методические указания предназначены для студентов, обучающихся по направлению подготовки «Программная инженерия» и профилю подготовки «Разработка программных продуктов и проектирование информационных систем» изучающих курс «Объектно-ориентированное программирование».

Рецензенты: к.т.н. А. В. Голышко

Литературный редактор Н. К. Костыгина
Изд. лицензия №020456 от 04.03.97
Подписано в печать 00.00.0000. Формат 60x84 1/16
Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,86. Усл.кр.-отт. 7,44. Уч.-изд. л. 2,0.
Тираж 100 экз. Заказ 00. Бесплатно

Московский технологический университет

117454, Москва, проспект Вернадского, 78

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1
ЦИКЛЫ, УСЛОВИЯ, ПЕРЕМЕННЫЕ И МАССИВЫ В JAVA.

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Целями данной лабораторной работы являются получение практических навыков разработки программ, изучение синтаксиса языка Java, освоение основных конструкций языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Язык Java - это объектно-ориентированный язык программирования. Программы написанные на Java могут выполняться на различных операционных системах при наличии необходимого ПО - Java Runtime Environment.

Для того чтобы создать программу на языке Java необходимо следующее ПО:

- Java Development Kit (JDK)
- Java Runtime Environment (JRE)
- Среда разработки. Например NetBeans или IDE IntelliJ IDEA.

Создание программы на Java.

Чтобы начать написание программы необходимо запустить среду разработки. При первом запуске среды обычно нужно указать путь к JDK, чтобы можно было компилировать код и запускать программу. В среде разработки необходимо создать Java проект, после чего необходимо создать пакет и в нем создать какой-либо класс. Также в свойствах проекта нужно указать класс, с которого будет начинаться запуск программы.

В классе, с которого будет начинаться запуск программы обязательно должен быть статический метод main(String[]), который принимает в качестве аргументов массив строк и не возвращает никакого значения.

Пример:

```
package example;

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

    }
}
```

В этом примере создан класс Example, располагающийся в пакете example. В нем содержится статический(ключевое слово static) метод main. Массив строк, который передается методу main() - это аргументы командной строки. При запуске Java программы, выполнение начнется с метода main().

Переменные.

Чтобы объявить переменную, необходимо указать тип переменной и ее имя. Типы переменной могут быть разные: целочисленный(long, int, short, byte), число с плавающей запятой(double, float), логический(boolean), перечисление, объектный(Object).

Переменным можно присваивать различные значения с помощью оператора присваивания "=".

Целочисленным переменным можно присваивать только целые числа, а числам с плавающей запятой - дробные. Целые числа обозначаются цифрами от 0 до 9, а дробные можно записывать отделяю целую часть от дробной с помощью точки. Переменным типа float необходимо приписывать справа букву "f", обозначающую, что данное число типа float. Без этой буквы число будет иметь тип double.

Класс String - особый класс в Java, так как ему можно присваивать значение, не создавая экземпляра класса(Java это сделает автоматически). Этот класс предназначен для представления строк. Строковое значение записывается буквами внутри двойных кавычек.

Пример:

```
float length = 2.5f;
double radius = 10024.5;
int meanOfLife = 42;
Object object = new String("Hello, world!");
String b = "Once compiled, runs everywhere?";
```

С целочисленными переменными можно совершать различные операции: сложение, вычитание, умножение, целое от деления, остаток от деления. Эти операции обозначаются соответственно "+", "-", "*", "/", "%". Для чисел с плавающей запятой применимы операции сложения, вычитания, умножения, деления. Для строк применима операция "+", обозначающая конкатенацию, слияние строк.

Массивы.

Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Для того чтобы создать массив переменных, необходимо указать квадратные скобки при объявлении переменной массива. После чего необходимо создать массив с помощью оператора new. Необходимо указать в квадратных скобках справа размер массива. Например, чтобы создать массив из десяти целочисленных переменных типа int, можно написать так:

```
int[] b = new int[10];
```

Для того чтобы узнать длину массива, необходимо обратиться к его свойству length через точку, например b.length.

Для того чтобы получить какой либо элемент массива, нужно указать после имени массива в квадратных скобках индекс, номер элемента. Массивы нумеруются с нуля. Например, чтобы получить 5 элемент массива, можно написать так: b[4].

Условия.

Условие - это конструкция, позволяющая выполнять то или другое действие, в зависимости от логического значения, указанного в условии. Синтаксис создания условия следующий:

```
if(a==b) {
   //Если a равно b, то будут выполняться операторы в этой области
} else {
   //Если a не равно b, то будут выполняться операторы в этой области
}
```

Если логическое условие, указанное в скобках после ключевого слова if, истинно, то будет выполняться блок кода, следующий за if, иначе будет выполняться код, следующий за ключевым словом else. Блок else не обязателен и может отсутствовать.

Скобками "{", "}" обозначается блок кода, который будет выполняться. Если в этом блоке всего 1 оператор, то скобки можно не писать(для условий и циклов).

Логическое условие составляется с помощью переменных и операторов равенства, неравенства, больше, меньше, больше или равно, меньше или равно, унарная операция не. Эти операторы обозначаются соответственно "==", "!=", ">", "<", ">=", "<=", "!". Результатом сравнения является логическое значение типа boolean, которое может иметь значение true ("истина") или false ("ложь"). Логические значения могут храниться в переменных типа boolean.

Циклы.

Цикл - это конструкция, позволяющая выполнять определенную часть кода несколько раз. В Java есть три типа циклов for, while, do while.

Цикл for - это цикл со счетчиком, обычно используется, когда известно, сколько раз должна выполниться определенная часть кода. Синтаксис цикла for:

```
for(int i=0;i<10;i++) {
    //Действия в цикле
}
```

В данном примере, в цикле объявлена переменная i, равная изначально 0. После точки с запятой ";" написано условие, при котором будет выполняться тело цикла (пока i<10), после второй точки с запятой указывается как будет изменяться переменная i (увеличиваться на 1 каждый раз с помощью операции инкремента "++"). Прописывать условие, объявлять переменную и указывать изменение переменной в цикле for не обязательно, но обязательно должны быть точки с запятой.

Цикл while - это такой цикл, который будет выполняться, пока логическое выражение, указанное в скобках истинно. Синтаксис цикла while:

```
while(logic) {
   //Тело цикла
}
```

В данном примере тело цикла будет выполняться, пока значение логической переменной logic равно true, то есть истинно.

Цикл do while - это такой цикл, тело которого выполнится хотя бы один раз. Тело выполнится более одного раза, если условие, указанное в скобках истинно.

```
do {
   //Тело цикла
}while(logic);
```

Потоки ввода/вывода и строки в Java, класс String

Для ввода данных используется класс Scanner из библиотеки пакетов

Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала открытого класса в коде программы.

В классе есть методы для чтения очередного символа заданного типа со стандартного потока ввода, а также для проверки существования такого символа.

Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) — уже знакомым вам объектом System.out. Есть ещё стандартный поток для вывода ошибок — System.err

```
importjava.util.Scanner; // импортируем  класс
importjava.util.Scanner; // импортируем  класс

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in); // создаём
        объект
класса  Scanner

        int i = 2;
        System.out.print("Введитецелоечисло: ");
        if(sc.hasNextInt()) { //
        возвращаетистиннуеслиспотока
ввода  можно  считать
целое   число
```

```
            i = sc.nextInt(); // считывает целое число
 с потока ввода  и  сохраняем  в   переменную

            System.out.println(i*2);

          } else {

            System.out.println("Вы  ввели  не  целое
            число");

          }

      }

       }
```

Имеется также метод nextLine(), позволяющий считывать целую последовательность символов, т.е. строку, а, значит, полученное через этот метод значение нужно сохранять в объекте класса String. В следующем примере создаётся два таких объекта, потом в них поочерёно записывается ввод пользователя, а далее на экран выводится одна строка, полученная объединением введённых последовательностейсимволов.

```
Importjava.util.Scanner;
public class Main {
    public  static  void  main(String[]
        args)  {  Scanner  sc  =  new
        Scanner(System.in);  String  s1,
        s2;
        s1 =
        sc.nextLine();
        s2 =
        sc.nextLine();
        System.out.println(s1 + s2);

    }

}
```

Существует и метод hasNext(), проверяющий остались ли в потоке ввода какие-то символы.

В классе String существует масса полезных методов, которые можно применять к строкам (перед именем метода будем указывать тип того значения, которое он возвращает):

- intlength() — возвращает длину строки (количество символов в ней);
- boolean isEmpty() — проверяет, пустая листрока;
- String replace(a, b) — возвращает строку, где символ a (литерал или переменная типа char) заменён на символb;
- String toLowerCase() — возвращает строку, где все символы исходной строки преобразованы кстрочным;
- String toUpperCase() — возвращает строку, где все символы исходной строки преобразованы кпрописным;
- boolean equals(s) — возвращает истинну, если строка к которой применён метод, совпадает со строкой s указанной в аргументе метода (с помощью оператора == строки сравнивать нельзя, как и любые другиеобъекты);
- int indexOf(ch) — возвращает индекс символа ch в строке (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). Если символ совсем не будет найден, то возвратит -1. Если символ встречается в строке несколько раз, то вовзвратит индекс его первоговхождения.
- int lastIndexOf(ch) — аналогичен предыдущему методу, но возвращает индекс последнего вхождения, если смивол встретился в строке несколькораз.
- int indexOf(ch,n) — возвращает индекс символа ch в строке, но начинает проверку с индекса n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). char charAt(n) — возвращает код символа, находящегосяв строке под индексом n (индекс это порядковый номер символа, но нумероваться символы начинают снуля).

```
public class Main {

    public static void main(String[] args) {
        String s1 = "firefox";

        System.out.println(s1.toUpperCase()); // выведет
«FIREFOX»
```

```
        String s2 = s1.replace('o', 'a');
        System.out.println(s2); // выведет «firefax»
        System.out.println(s2.charAt(1)); // выведет «i»
        int i;
        i = s1.length();
        System.out.println(i); // выведет  7
        i = s1.indexOf('f');
        System.out.println(i); // выведет  0
        i = s1.indexOf('r');
        System.out.println(i); // выведет  2
        i = s1.lastIndexOf('f');
        System.out.println(i); // выведет  4
        i = s1.indexOf('t');
        System.out.println(i); // выведет -1
        i = s1.indexOf('r',3);
        System.out.println(i); // выведет -1
    }
}
```

Пример программы, которая выведет на экран индексы всех пробелов в строке, введенноё пользователем с клавиатуры:
importjava.util.Scanner;

```
public class Main {

    public static void main(String[] args) {
        Scanner sc = new
        Scanner(System.in); String s =
        sc.nextLine();
        for(int i=0; i < s.length();
            i++) { if(s.charAt(i) == ' ')
            {
                System.out.println(i);
            }
        }
    }
}
```

Методы в языке Java.

Методы позволяют выполнять блок кода, из любого другого места, где это доступно. Методы определяются внутри классов. Методы могут

быть статическими(можно выполнять без создания экземпляра класса), не статическими (не могут выполняться без создания экземпляра класса). Методы могут быть открытыми(public), закрытыми(private). Закрытые методы могут вызываться только внутри того класса, в котором они определены. Открытые методы можно вызывать для объекта внутри других классов.

При определении метода можно указать модификатор доступа(public, private, protected), а также указать статический ли метод ключевым словом static. Нужно обязательно указать тип возвращаемого значения и имя метода. В скобках можно указать аргументы, которые необходимо передать методу для его вызова. В методе с непустым типом возвращаемого значения нужно обязательно указать оператор return и значение, которое он возвращает. Если метод не возвращает никакого значения, то указывается тип void.

Пример:

```
public static int sum(int a, int b) {
   return a+b;
}
```

В данном примере определен метод, возвращающий сумму двух чисел a и b. Этот метод статический, и его можно вызывать не создавая экземпляра класса, в котором он определен.

Чтобы вызвать этот метод внутри класса, в котором он создан необходимо написать имя метода и передать ему аргументы. Пример:

```
int s = sum(10,15);
```

ВАРИАНТЫ ЗАДАНИЙ

1. Вывести на экран сумму чисел массива с помощью циклов for, while, do while.

2. Вывести на экран аргументы командной строки в цикле for.

3. Вывести на экран первые 10 чисел гармонического ряда.

4. Сгенерировать массив целых чисел случайным образом, вывести его на экран, отсортировать его, и снова вывести на экран.

5. Создать метод, вычисляющую факториал числа с помощью цикла, проверить работу метода.

ЛАБОРАТОРНАЯ РАБОТА №2
ООП В JAVA. ПОНЯТИЕ КЛАССА.

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы - изучить основные концепции объектно-ориентированного программирования, изучить понятие класса и научиться создавать классы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Язык Java - объектно-ориентированный язык программирования. В центре ООП находится понятие объекта. Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Класс в ООП — это в чистом виде абстрактный тип данных, создаваемый программистом. С этой точки зрения объекты являются значениями данного абстрактного типа, а определение класса задаёт внутреннюю структуру значений и набор операций, которые над этими значениями могут быть выполнены. Желательность иерархии классов (а значит, наследования) вытекает из требований к повторному использованию кода — если несколько классов имеют сходное поведение, нет смысла дублировать их описание, лучше выделить общую часть в общий родительский класс, а в описании самих этих классов оставить только различающиеся элементы.

Необходимость совместного использования объектов разных классов, способных обрабатывать однотипные сообщения, требует

поддержки полиморфизма — возможности записывать разные объекты в переменные одного и того же типа. В таких условиях объект, отправляя сообщение, может не знать в точности, к какому классу относится адресат, и одни и те же сообщения, отправленные переменным одного типа, содержащим объекты разных классов, вызовут различную реакцию.

Создание классов в Java.

Для того чтобы создать класс в языке Java необходимо создать файл с расширением java. Имя файла должно быть таким же, как и имя создаваемого класса. В созданном файле должен описываться класс. Синтаксис написания класса:

```
<модификатор досупа> class <имя класса> {
   <тело класса>
}
```

В качестве модификатора доступа можно указать ключевое слово public или private. Если указано слово public, то класс будет доступен из других пакетов. Если указано слово private, то класс будет доступен только внутри того пакета, в котором он находится.

В теле класса можно описать методы, переменные, константы, конструкторы класса.

Конструктор - это специальный метод, который вызывается при создании нового объекта. Не всегда удобно инициализировать все переменные класса при создании его экземпляра. Иногда проще, чтобы какие-то значения были бы созданы по умолчанию при создании объекта. По сути конструктор нужен для автоматической инициализации переменных.

Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.

В отличие от метода, конструктор никогда ничего не возвращает.

Пример класса, описывающего прямоугольник с высотой height и шириной width.

```
public class Rectangle {
   //Свойства, поля класса
   private float width;
   private float height;
```

```
    //Конструктор класса
    public Rectangle(float w, float h) {
        width=w;
        height=h;
    }

    //Метод, возвращающий ширину прямоугольника
    public float getWidth() {
        return width;
    }

    //Метод, возвращающий высоту прямоугольника
    public float getHeight() {
        return height;
    }

    //Метод, устанавливающий ширину прямоугольника
    public void setWidth(float w) {
        width=w;
    }

    //Метод, устанавливающий высоту прямоугольника
    public void setHeight(float h) {
        height=h;
    }

}
```

В данном примере был создан класс с одним конструктором, и методами, меняющими поля класса setWidth(), setHeight() ("сеттеры"), и возвращающие их значение getWidth(), getHeight() ("геттеры").

Если конструкторы в классе отсутствуют, то Java автоматически создает конструктор по умолчанию, который не имеет аргументов.

Создание экземпляра класса.

Для того чтобы создать экземпляр класса необходимо объявить переменную, тип которой соответствует имени класса или суперкласса. После чего нужно присвоить этой переменной значение,

вызвав конструктор создаваемого класса с помощью оператора new. Например, можно создать экземпляр класса Rectangle следующим образом:

```
Rectangle rect = new Rectangle(20, 10);
```

После этого можно вызывать методы этого класса для объекта rect, указав имя метода через точку:

```
rect.setWidth(20);
System.out.println("Новая ширина: "+rect.getWidth());
```

ВАРИАНТЫ ЗАДАНИЙ

1. Создать класс, описывающий модель окружности (Circle). В классе должны быть описаны нужные свойства окружности и методы для получения, изменения этих свойств. Протестировать работу класса в классе CircleTest, содержащим метод статический main(String[] args).

2. Создать класс, описывающий тело человека(Human). Для описания каждой части тела создать отдельные классы(Head, Leg, Hand). Описать необходимые свойства и методы для каждого класса. Протестировать работу класса Human.

3. Создать класс, описывающий книгу (Book). В классе должны быть описаны нужные свойства книги(автор, название, год написания и т. д.)и методы для получения, изменения этих свойств. Протестировать работу класса в классе BookTest, содержащим метод статический main(String[] args).

ЛАБОРАТОРНАЯ РАБОТА №3
НАСЛЕДОВАНИЕ В JAVA

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы - изучить понятие наследования, и научиться реализовывать наследование в Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

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:

```
public class Person {

    private String name;
    private String surname;

    public String getName() { return name; }
    public String getSurname() { return surname; }

    public Person(String name, String surname){

        this.name=name;
        this.surname=surname;
    }

    public void displayInfo(){

        System.out.println("Имя: " + name + " Фамилия: " + surname);
    }
}
```

И, возможно, впоследствии мы решили расширить имеющуюся систему и классов, добавив в нее класс, описывающий сотрудника предприятия - класс Employee. Так как этот класс

реализует тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником) от класса Person, который, в свою очередь, называется базовым классом или родителем:

```
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.

В классе Employee могут быть определены свои методы и поля, а также конструктор. Способность к изменению функциональности, унаследованной от базового класса, назвывается полиморфизмом и является одинм из ключевых аспектов объектно-ориентированного программирования наряду с наследованием и инкапсуляцией.

Например, переопределим метод displayInfo() класса Person в классе Employee:

```
class Employee extends Person{

    private String company;

    public Employee(String name, String surname, String company) {

        super(name, surname);
        this.company=company;
    }

    public void displayInfo(){

        super.displayInfo();
        System.out.println("Компания: " + company);
    }
}
```

Класс Employee определяет дополнительное поле для хранения компании, в которой работает сотрудник. Кроме того, оно также устанавливается в конструкторе.

Так как поля name и surname в базовом классе Person объявлены с модификатором private, то мы не можем к ним напрямую обратиться из класса Employee. Однако в данном случае нам это не нужно. Чтобы их установить, мы обращаемся к конструктору базового класса с помощью ключевого слова super, в скобках после которого идет перечисление передаваемых аргументов.

С помощью ключевого слова super мы можем обратиться к любому члену базового класса - методу или полю, если они не определены с модификатором private.

Также в классе Employee переопределяется метод displayInfo() базового класса. В нем с помощью ключевого super также идет обращение к методу displayInfo(), но уже базового класса, и затем выводится дополнительная информация, относящаяся только к Employee.

Используя обращение к методом базового класса, можно было бы переопределить метод displayInfo() следующим образом:

```
public void displayInfo(){

    System.out.println("Имя: " + super.getName() + " Фамилия: "
        + super.getSurname() + " Компания: " + company);
}
```

При этом нам необязательно переопределять все методы базового класса. Например, в данном случае мы не переопределяем методы getName() и getSurname(). Поэтому для этих методов класс-наследник будет использовать реализацию из базового класса. И в основной программе мы можем эти методы использовать:

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

    Employee empl = new Employee("Tom", "Simpson", "Oracle");
    empl.displayInfo();
    String firstName = empl.getName();
    System.out.println(firstName);
```

```
}
```

Хотя наследование очень интересный и эффективный механизм, но в некоторых ситуациях его применение может быть нежелательным. И 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 displayInfo(), запретим его переопределение:

```
public class Person {

    //.....................

    public final void displayInfo(){

        System.out.println("Имя: " + name + " Фамилия: " + surname);
    }
}
```

В этом случае в классе Employee надо будет создать метод с другим именем для вывода информации об объекте.

Абстрактные классы

Кроме обычных классов в Java есть абстрактные классы. Абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. А производные классы уже реализуют этот функционал.

При определении абстрактных классов используется ключевое слово abstract:

```
public abstract class Human{

    private int height;
    private double weight;

    public int getHeight() { return height; }
    public double getWeight() { return weight; }
}
```

Кроме обычных методов абстрактный класс может содержать абстрактные методы. Такие методы определяются с помощью ключевого слова abstract и не имеют никакого функционала:

```
public abstract void displayInfo();
```

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

Зачем нужны абстрактные классы? Допустим, мы делаем программу для обсулживания банковских операций и определяем в ней три класса: Person, который описывает человека, Employee, который описывает банковского служащего, и класс Client, который представляет клиента банка. Очевидно, что классы Employee и Client будут производными от класса Person, так как оба класса имеют некоторые общие поля и методы. И так как все объекты будут представлять либо сотрудника, либо клиента банка, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным.

```
public abstract class Person {

    private String name;
    private String surname;

    public String getName() { return name; }
    public String getSurname() { return surname; }

    public Person(String name, String surname){
```

```
        this.name=name;
        this.surname=surname;
    }

    public abstract void displayInfo();
}

class Employee extends Person{

    private String bank;

    public Employee(String name, String surname, String company) {

        super(name, surname);
        this.bank=company;
    }

    public void displayInfo(){

        System.out.println("Имя: " + super.getName() + " Фамилия: "
                + super.getSurname() + " Работает в банке: " + bank);
    }
}

class Client extends Person
{
    private String bank;

    public Client(String name, String surname, String company) {

        super(name, surname);
        this.bank=company;
    }

    public void displayInfo(){

        System.out.println("Имя: " + super.getName() + " Фамилия: "
                + super.getSurname() + " Клиент банка: " + bank);
    }
```

```
}
```

ВАРИАНТЫ ЗАДАНИЙ

1. Создать абстрактный класс, описывающий посуду(Dish). С помощью наследования реализовать различные виды посуды, имеющие свои свойства и методы. Протестировать работу классов.

2. Создать абстрактный класс, описывающий собак(Dog). С помощью наследования реализовать различные породы собак. Протестировать работу классов.

3. Создать абстрактный класс, описывающий мебель. С помощью наследования реализовать различные виды мебели. Также создать класс FurnitureShop, моделирующий магазин мебели. Протестировать работу классов.

ЛАБОРАТОРНАЯ РАБОТА №4
ИНТЕРФЕЙСЫ В JAVA.

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы - изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

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

Чтобы определить интерфейс, используется ключевое слово interface.

Определим следующий интерфейс:

```
public interface Printable{

    void print();
}
```

Интерфейс может определять различные методы, которые, так же как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

И также при объявлении интерфейса надо учитывать, что только один интерфейс в файле может иметь тип доступа public. А его название

должно совпадать с именем файла. Остальные интерфейсы (если такие имеются в файле java) не должны иметь модификаторов доступа.

Интерфейс может определять различные методы, которые, так же как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

И также при объявлении интерфейса надо учитывать, что только один интерфейс в файле может иметь тип доступа public. А его название должно совпадать с именем файла. Остальные интерфейсы (если такие имеются в файле java) не должны иметь модификаторов доступа.

Чтобы класс применил интерфейс, надо использовать ключевое слово implements:

```
class Book implements Printable{

    String name;
    String author;
    int year;

    Book(String name, String author, int year){
        this.name = name;
        this.author = author;
        this.year = year;
    }

    public void print() {

        System.out.printf("Книга '%s' (автор %s) была издана в %d году \n",
name, author, year);
    }
}
```

При этом надо учитывать, что если класс применяет интерфейс, то он должен реализовать все методы интерфейса, как в случае выше реализован метод print.

Потом в главном классе мы можем использовать данный класс и его метод print:

```
Book b1 = new Book("Война и мир", "Л. Н. Толстой", 1863);
b1.print();
```

В тоже время мы не можем напрямую создавать объекты интерфейсов, поэтому следующий код не будет работать:

```
Printable pr = new Printable();
pr.print();
```

Одним из преимуществ использоваия интерфейсов является то, что они позволяют добавить в приложение гибкости. Например, в дополнение к классу Book определим еще один класс, который будет реализовывать интерфейс Printable:

```
public class Journal implements Printable {

    private String name;

    String getName(){
        return name;
    }

    Journal(String name){

        this.name = name;
    }
    public void print() {
        System.out.printf("Журнал '%s'\n", name);
    }
}
```

Класс Book и класс Journal связаны тем, что они реализуют интерфейс Printable. Поэтому мы динамически в программе можем создавать объекты Printable как экземпляры обоих классов:

```
Printable printable = new Book("Война и мир", "Л. Н. Толстой", 1863);
printable.print();
printable = new Journal("Хакер");
printable.print();
```

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

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

    Printable printable = createPrintable("Компьютерра",false);
    printable.print();

    read(new Book("Отцы и дети", "И. Тургенев", 1862));
    read(new Journal("Хакер"));
}

static void read(Printable p){

    p.print();
}

static Printable createPrintable(String name, boolean option){

    if(option)
        return new Book(name, "неизвестен", 2015);
    else
        return new Journal(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 вожем возвратить как объект Book, так и Journal.

ВАРИАНТЫ ЗАДАНИЙ

1. Создать интерфейс Nameable, с методом getName(), возвращающим имя объекта, реализующего интерфейс. Проверить работу для различных объектов (например, можно создать классы, описывающие разные сущности, которые могут иметь имя: планеты, машины, животные и т. д.).
2. Реализовать интерфейс Priceable, имеющий метод getPrice(),

возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

ЛАБОРАТОРНАЯ РАБОТА №5

СОЗДАНИЕ ПРОГРАМ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ ПОЛЬЗОВАТЕЛЯ НА JAVA

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы - научиться создавать графический интерфейс пользователя, освоить на практике работу с различными объектами для создания ГИП, менеджерами размещения компонентов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Для создания графического интерфейса пользователя можно использовать стандартную Java библиотеку Swing или AWT. В этих библиотеках имеются различные классы, позволяющие создавать окна, кнопки, текстовые поля, меню и другие объекты.

Text Fields - текстовое поле или поля для ввода текста (можно ввести только одну строку). Примерами текстовых полей являются поля для ввода логина и пароля, например, используемые, при входе в электронную почту.

Пример содания объекта класса JTextField:

```
JTextField jta = new JTextField (10);
```

В параметре конструктора задано число 10, это количество символов, которые могут быть видны в текстовом поле. Текст веденный в поле JText может быть возвращен с помощью метода getText(). Также в поле можно записать новое значение с помощью метода setText(String s).

Как и у других компонентов, мы можем изменять цвет и шрифт текста в текстовом поле.

```
class LabExample extends JFrame
{
    JTextField jta = new JTextField(10);
    Font fnt = new Font("Times new
```

```
     roman",Font.BOLD,20); LabExample()
     {
          super("Example");
          setLayout(new
          FlowLayout());
          setSize(250,100);
          add(jta);
          jta.setForeground(Color.PI
          NK); jta.setFont(fnt);
          setVisible(true);
     }

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

Важное замечание

Ответственность за выполнение проверки на наличие ошибок в коде лежит полностью на программисте, например, чтобы проверить произойдет ли ошибка, когда в качестве входных данных в JTextField ожидается ввод числа. Компилятор не будет ловить такого рода ошибку, поэтому ее необходимо обрабатывать пользовательским кодом.

Выполните следующий пример и наблюдайте за результатом, когда число вводится в неправильном формате:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class LabExample extends JFrame
{
     JTextField jta1 = new
```

```
    JTextField(10); JTextField jta2
    = new JTextField(10);
    JButton button = new JButton(" Add them
    up");
    Font fnt = new Font("Times new
    roman",Font.BOLD,20); LabExample()
    {
        super("Example");
        setLayout(new
        FlowLayout());
        setSize(250,150);
        add(new JLabel("1st
        Number")); add(jta1);
        add(new JLabel("2nd
        Number")); add(jta2);
        add(button);

        button.addActionListener(new
        ActionListener()
        {
            public void actionPerformed(ActionEvent
            ae)
            {
                try
                {
                    double x1 =
Double.parseDouble(jta1.getText().tr
im());
                    double x2 =
Double.parseDouble(jta2.getText().tr
im());

                    JOptionPane.showMessageDialog(n
ull, "Result =
"+(x1+x2),"Alert",JOptionPane.INFORMATION_MESSAGE);

                }
                catch(Exception e)
                {
                     JOptionPane.showMessageDialog(
null, "Error in Numbers !","alert" ,
JOptionPane.ERROR_MESSAGE);
                }
```

```
                }
            });

            setVisible(true);
        }

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

JTextArea

Компонент TextAreas похож на TextFields, но в него можно вводить более одной строки. В качестве примера TextArea можно рассмотреть текст, который мы набираем в теле сообщения электронной почты

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class TextAreaExample extends JFrame
{
    JTextArea jta1 = new JTextArea(10,25);
    JButton button = new JButton("Add some Text");
    public TextAreaExample()
    {
        super("Example");
        setSize(300,300);
         setLayout(new
         FlowLayout());
         add(jta1);
         add(button);
         button.addActionListener(new
         ActionListener()
         {
             public void actionPerformed(ActionEvent
             ae)
             {
                 String txt =
JOptionPane.showInputDialog(null,"Insert some
```

```
text");
                    jta1.append(txt);
                }
            });
        }
        public static void main(String[]args)
        {
            new TextAreaExample().setVisible(true);
        }
}
```

Замечание

Мы можем легко добавить возможность прокрутки к текстовому полю, добавив его в контейнер с именем JScrollPane следующим образом:

```
JTextArea txtArea = new JTextArea(20,20)
JScrollPane jScroll = new
JScrollPane(txtArea);
// …
add(Scroll); // we add the scrollPane and not the text
area.
```

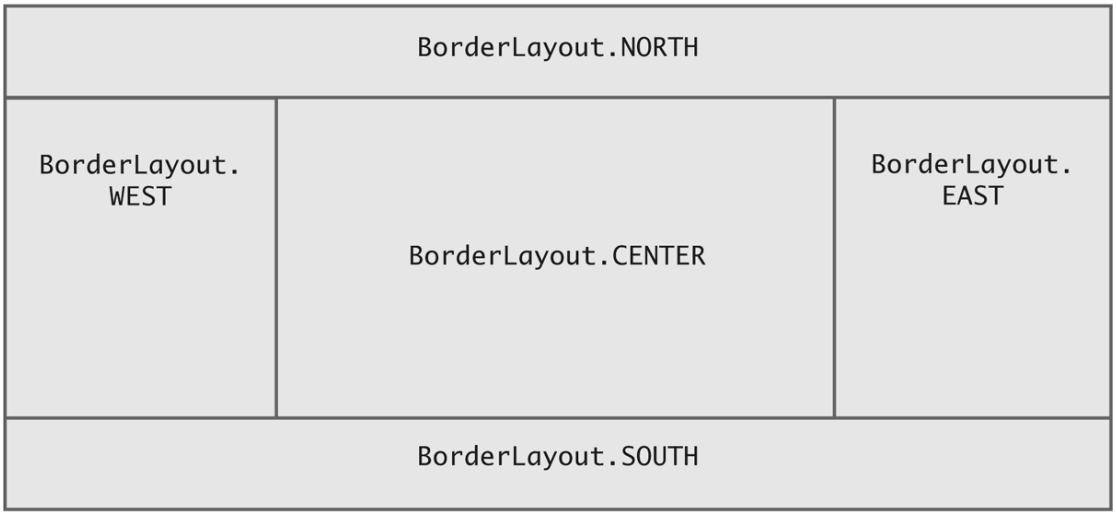
Попробуйте выполнить сами!

Layout Менеджеры:

BorderLayout:

Разделяеткомпонентнапятьобластей (WEST, EAST, NOTH, SOUTH and Center). Другин компоненты могут быть добавлены в любой из этих компонентов пятерками.

Display 17.8 **BorderLayout Regions**



Метод для добавления в контейнер, который есть у менеджера BorderLayout отличается и выглядит следующим образом:

```
add( comp , BorderLayout.EAST);
```

Обратите внимание, что мы можем например добавить панели JPanel в эти области и затем добавлять компоненты этих панелей. Мы можем установить расположение этих JPanel используя другие менеджеры

GridLayout менеджер

С помощью менеджера GridLayout компонент может принимать форму таблицы, где можно задать число строк и столбцов.

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

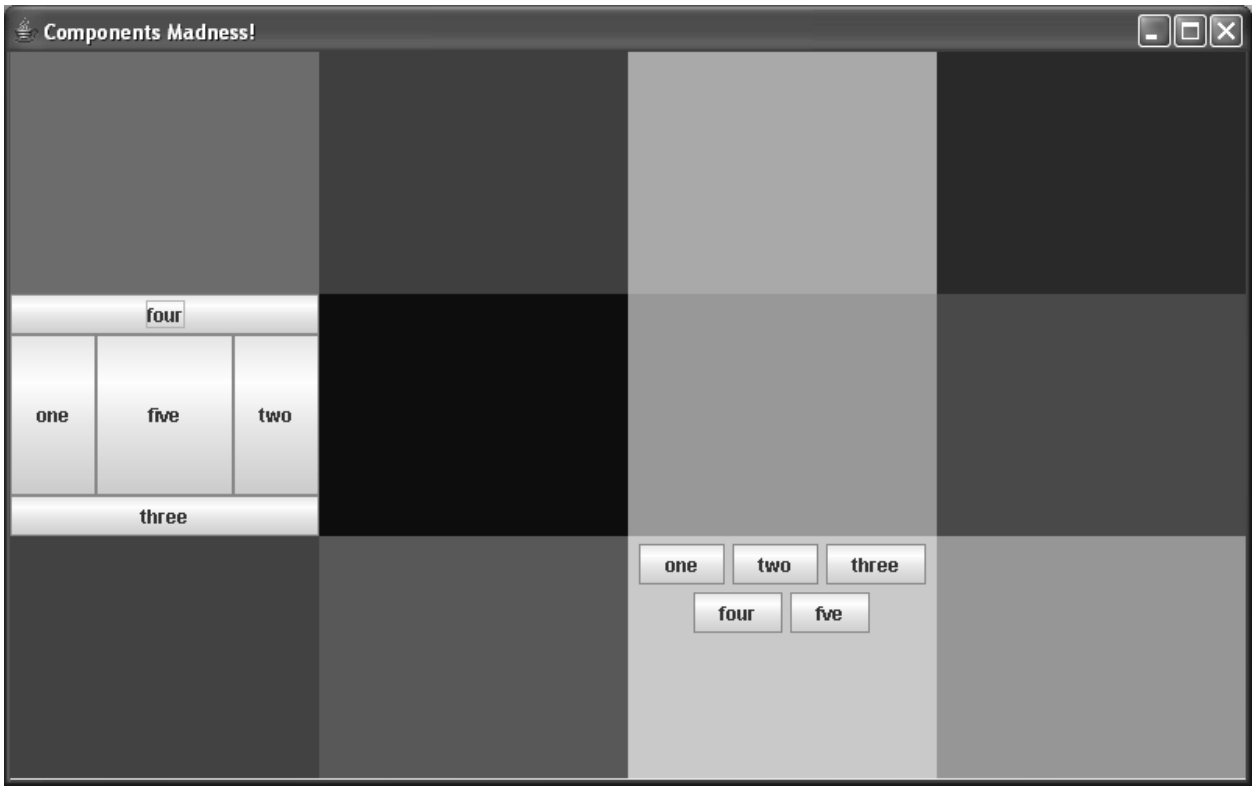
Если компоненту GridLayout задать 3 строки и 4 столбца, то компоненты будут принимать форму таблицы, показанной выше, и будут всегда будут добавляться в порядке их появления.

Следующий пример иллюстрирует смесь компоновки различных компонентов

```
import
javax.swing.*;
import java.awt.*;
importjava.awt.event
.*;


class BorderExample extends JFrame
{
    JPanel[] pnl = new JPanel[12];

    public BorderExample()
    {
        setLayout(new GridLayout(3,4));
        for(int i = 0 ; i < pnl.length ;
        i++)
        {
            int r = (int) (Math.random() *
            255); int b = (int)
            (Math.random() * 255); int g =
            (int) (Math.random() * 255);
            pnl[i] = new JPanel();
            pnl[i].setBackground(new
            Color(r,g,b)); add(pnl[i]);
        }

        pnl[4].setLayout(new BorderLayout());
        pnl[4].add(new
        JButton("one"),BorderLayout.WEST);
        pnl[4].add(new
        JButton("two"),BorderLayout.EAST);
        pnl[4].add(new
        JButton("three"),BorderLayout.SOUTH);
        pnl[4].add(new
        JButton("four"),BorderLayout.NORTH);
```

```
            pnl[4].add(new
            JButton("five"),BorderLayout.CENTER);

            pnl[10].setLayout(new
            FlowLayout()); pnl[10].add(new
            JButton("one"));
            pnl[10].add(new JButton("two"));
            pnl[10].add(new JButton("three"));
            pnl[10].add(new JButton("four"));
            pnl[10].add(new JButton("fve"));

            setSize(800,500);
      }
      public static void main(String[]args)
      {
            new BorderExample().setVisible(true);
      }
}
```

Вот такой будет иметь вид, представленный выше код



Заметьте, что JFrame имеет GridLayout размера 3 на 4 (таблица), в то время как JPanelразмером (2, 1) имеет менеджер BorderLayout. A JPanel (3, 3) имеет FLowLayout.

Null Layout Manager

Иногда бывает нужно изменить размер и расположение компонента в контейнере. Таким образом, мы должны указать программе не использовать никакой менеджер компоновки, то есть

(setLayout (нуль)). Так что мы получим что-то вроде этого:

```
import
javax.swing.*;
import java.awt.*;
importjava.awt.event
.*;

class NullLayout extends JFrame
{
     JButton but1 = new
     JButton("One");; JButton but2 =
     new JButton("two");; JButton
     but3 = new JButton("three");;

     public NullLayout()
     {
          setLayout(null);
          but1.setBounds(150,300,100,20); // added at
150,300 width = 100, height=20
          but2.setSize(80,400); // added at 0,0
```

```
width = 80, height=400
         but3.setLocation(300,1
         00);
         but3.setSize(200,75);
         // those two steps can be combined in one
setBounds method call
         add(but1);
         add(but2);
         add(but3);
         setSize(500,50
         0);
     }
     public static void main(String[]args)
     {
          new NullLayout().setVisible(true);
     }
}
```

Меню

Добавление меню в программе Java проста. Java определяет три компонента для обработки этих

- JMenuBar: который представляет собой компонент, который

содержит меню.

- JMenu: который представляет меню элементов для выбора.
- JMenuItem: представляет собой элемент, который можно кликнуть из меню.

Подобно компоненту Button (на самом деле MenuItems являются подклассами класса AbstractButton). Мы можем добавить ActionListener к ним так же, как мы делали с кнопками

ВАРИАНТЫ ЗАДАНИЙ

1. Создать окно, нарисовать в нем 20 случайных фигур, случайного цвета. Классы фигур должны наследоваться от абстрактного класса Shape, в котором описаны свойства фигуры: цвет, позиция.
2. Создать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки.
3. Создать окно, реализовать анимацию, с помощью картинки, состоящей из нескольких кадров.

ЛАБОРАТОРНАЯ РАБОТА №6

ОБРАБОТКА СОБЫТИЙ В JAVA ПРОГРАММАХ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ ПОЛЬЗОВАТЕЛЯ

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы - научиться обрабатывать различные события для разных компонентов(кнопок, меню и т. д.).

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Механизм обработки событий библиотеки Swing.

В контексте графического интерфейса пользователя наблюдаемыми объектами являются элементы управления: кнопки, флажки, меню и т.д. Они могут сообщить своим наблюдателям об определенных событиях, как элементарных (наведение мышкой, нажатие клавиши на клавиатуре), так и о высокоуровневых (изменение текста в текстовом поле, выбор нового элемента в выпадающем списке и т.д.).

Наблюдателями должны являться объекты классов, поддерживающих специальные интерфейсы (в классе наблюдателя должны быть определенные методы, о которых «знает» наблюдаемый и вызывает их при наступлении события). Такие классы в терминологии Swing называются слушателями.

Интерфейс MouseListener и обработка событий от мыши.

События от мыши — один из самых популярных типов событий. Практически любой элемент управления способен сообщить о том, что на него навели мышь, щелкнули по нему и т.д. Об этом будут оповещены все зарегистрированные слушатели событий от мыши.

Слушатель событий от мыши должен реализовать интерфейс MouseListener. В этом интерфейсе перечислены следующие методы:

- public void mouseClicked(MouseEvent event) — выполнен щелчок мышкой на наблюдаемом объекте
- public void mouseEntered(MouseEvent event) — курсор мыши вошел в область наблюдаемого объекта
- public void mouseExited(MouseEvent event) — курсор мыши вышел из области наблюдаемого объекта
- public void mousePressed(MouseEvent event) — кнопка мыши нажата в момент, когда курсор находится над наблюдаемым

объектом

- public void mouseReleased(MouseEvent event) — кнопка мыши отпущена в момент, когда курсор находится над наблюдаемым объектом

Чтобы обработать нажатие на кнопку, требуется описать класс, реализующий интерфейс MouseListener. Далее необходимо создать объект этого класса и зарегистрировать его как слушателя интересующей нас кнопки. Для регистрации слушателя используется метод addMouseListener().

Опишем класс слушателя в пределах класса окна SimpleWindow, после конструктора. Обработчик события будет проверять, ввел ли пользователь логин «Иван» и выводить сообщение об успехе или неуспехе входа в систему:

```
class MouseL implements MouseListener {

   public void mouseClicked(MouseEvent event) {
      if (loginField.getText().equals("Иван"))
         JOptionPane.showMessageDialog(null, "Вход выполнен");
      else
         JOptionPane.showMessageDialog(null, "Вход НЕ выполнен");
}

   public void mouseEntered(MouseEvent event) {}

   public void mouseExited(MouseEvent event) {}

   public void mousePressed(MouseEvent event) {}

   public void mouseReleased(MouseEvent event) {}

}
```

Мы сделали слушателя вложенным классом класса SimpleWindow, чтобы он мог легко получить доступ к его внутренним полям loginField и passwordField. Кроме того, хотя реально мы обрабатываем только одно из пяти возможных событий мыши, описывать пришлось все пять методов (четыре имеют пустую

реализацию). Дело в том, что в противном случае класс пришлось бы объявить абстрактным (ведь он унаследовал от интерфейса пустые заголовки методов) и мы не смогли бы создать объект этого класса. А мы должны создать объект слушателя и прикрепить его к кнопке. Для этого в код конструктора SimpleWindow() необходимо добавить команду:

ok.addMouseListener(new MouseL());

Это можно сделать сразу после команды:

JButton ok = new JButton("OK");

Создание слушателей с помощью анонимных классов

Чтобы кнопка ok обрела слушателя, который будет обрабатывать нажатие на нее, нам понадобилось описать новый (вложенный) класс. Иногда вместо вложенного класса можно обойтись анонимным. Анонимный класс не имеет имени и в программе может быть создан только один объект этого класса (создание которого совмещено с определением класса). Но очень часто слушатель пишется для того, чтобы обрабатывать события единственного объекта — в нашем случае кнопки ok, а значит, используется в программе только однажды: во время привязки к этому объекту. Таким образом, мы можем заменить вложенный класс анонимным. Для этого описание класса MouseL можно просто удалить, а команду

ok.addMouseListener(new MouseL());

заменить на:

```
ok.addMouseListener(new MouseListener() {

   public void mouseClicked(MouseEvent event) {
      if (loginField.getText().equals("Иван"))
         JOptionPane.showMessageDialog(null, "Вход выполнен");
      else JOptionPane.showMessageDialog(null, "Вход НЕ выполнен");
   }
   public void mouseEntered(MouseEvent event) {}
   public void mouseExited(MouseEvent event) {}
   public void mousePressed(MouseEvent event) {}
   public void mouseReleased(MouseEvent event) {}

});
```

Новый вариант выглядит более громоздко, чем первый. Злоупотребление анонимными классами может сделать программу плохо читаемой. Однако в результате все действия с кнопкой (создание, настройка ее внешнего вида и команды обработки щелчка по ней) не разнесены, как в случае вложенных классов, а находятся рядом, что облегчает сопровождение (внесение изменений) программы. В случае простых (в несколько строк) обработчиков разумно делать выбор в пользу анонимных классов.

Класс MouseAdapter

Программа стала выглядеть загроможденной главным образом из-за того, что помимо полезного для нас метода mouseClicked() пришлось определять пустые реализации всех остальных, не нужных методов. Но этого можно избежать.

Класс MouseAdapter реализует интерфейс MouseListener, определяя пустые реализации для каждого из его методов. Можно унаследовать своего слушателя от этого класса и переопределить те методы, которые нам нужны.

В результате предыдущее описание слушателя будет выглядеть более компактно:

```
ok.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent event) {
         if (loginField.getText().equals("Иван"))
            JOptionPane.showMessageDialog(null, "Вход выполнен");
      else JOptionPane.showMessageDialog(null, "Вход НЕ выполнен");
   }
});
```

Общая структура слушателей

Кроме слушателей MouseListener визуальные компоненты Swing поддерживают целый ряд других слушателей.

Каждый слушатель должен реализовывать интерфейс ***Listener, где *** — тип слушателя. Практически каждому из этих интерфейсов (за исключением тех, в которых всего один метод) соответствует пустой класс-заглушка ***Adapter. Каждый метод интерфейса слушателя принимает один параметр типа ***Event, в котором собрана вся информация, относящаяся к событию.

Чтобы привязать слушателя к объекту (который поддерживает

соответствующий тип слушателей) используется метод add***Listener(***Listener listener).

Например, слушатель MouseListener должен реализовать интерфейс с таким же именем, которому соответствует класс-заглушка MouseAdapter. Методы этого интерфейса обрабатывают параметр типа MouseEvent, а регистрируется слушатель методом addMouseListener(MouseListener listener).

Слушатель фокуса FocusListener

Слушатель FocusListener отслеживает моменты, когда объект получает фокус (то есть становится активным) или теряет его. Концепция фокуса очень важна для оконных приложений. В каждый момент времени в окне может быть только один активный (находящийся в фокусе) объект, который получает информацию о нажатых на клавиатуре клавишах (т.е. реагирует на события клавиатуры), о прокрутке колесика мышки и т.д. Пользователь активирует один из элементов управления нажатием мышки или с помощью клавиши Tab (переключаясь между ними).

Интерфейс FocusListener имеет два метода:

public void focusGained(FocusEvent event) — вызывается, когда наблюдаемый объект получает фокус

public void focusLost(FocusEvent event) — вызывается, когда наблюдаемый объект теряет фокус.

Слушатель колесика мышки MouseWheelListener

Слушатель MouseWheelListener оповещается при вращении колесика мыши в тот момент, когда данный компонент находится в фокусе. Этот интерфейс содержит всего один метод:

public void mouseWheelMoved(MouseWheelEvent event).

Слушатель клавиатуры KeyListener

Слушатель KeyListener оповещается, когда пользователь работает с клавиатурой в тот момент, когда данный компонент находится в фокусе. В интерфейсе определены методы:

public void mouseKeyTyped(KeyEvent event) — вызывается, когда с клавиатуры вводится символ

public void mouseKeyPressed(KeyEvent event) — вызывается, когда нажата клавиша клавиатуры

public void mouseKeyReleased(KeyEvent event) — вызывается, когда отпущена клавиша клавиатуры.

Аргумент event этих методов способен дать весьма ценные сведения. В частности, команда event.getKeyChar() возвращает символ типа char, связанный с нажатой клавишей. Если с нажатой клавишей не связан никакой символ, возвращается константа CHAR_UNDEFINED. Команда event.getKeyCode() возвратит код нажатой клавиши в виде целого числа типа int. Его можно сравнить с одной из многочисленных констант, определенных в классе KeyEvent: VK_F1, VK_SHIFT, VK_D, VK_MINUS и т.д. Методы isAltDown(), isControlDown(), isShiftDown() позволяют узнать, не была ли одновременно нажата одна из клавиш-модификаторов Alt, Ctrl или Shift.

Слушатель изменения состояния ChangeListener

Слушатель ChangeListener реагирует на изменение состояния объекта. Каждый элемент управления по своему определяет понятие «изменение состояния». Например, для панели со вкладками JTabbedPane это переход на другую вкладку, для ползунка JSlider — изменение его положения, кнопка JButton рассматривает как смену состояния щелчок на ней. Таким образом, хотя событие это достаточно общее, необходимо уточнять его специфику для каждого конкретного компонента. В интерфейсе определен всего один метод:

public void stateChanged(ChangeEvent event).

Слушатель событий окна WindowListener

Слушатель WindowListener может быть привязан только к окну и оповещается о различных событиях, произошедших с окном:

public void windowOpened(WindowEvent event) — окно открылось.

public void windowClosing(WindowEvent event) — попытка закрытия окна (например, пользователя нажал на крестик). Слово «попытка» означает, что данный метод вызовется до того, как окно будет закрыто и может воспрепятствовать этому (например, вывести диалог типа «Вы уверены?» и отменить закрытие окна, если пользователь выберет «Нет»).

public void windowClosed(WindowEvent event) — окно закрылось.

public void windowIconified(WindowEvent event) — окно свернуто.

public void windowDeiconified(WindowEvent event) — окно развернуто.

public void windowActivated(WindowEvent event) — окно стало активным.

public void windowDeactivated(WindowEvent event) — окно стало неактивным.

Слушатель событий компонента ComponentListener

Слушатель ComponentListener оповещается, когда наблюдаемый визуальный компонент изменяет свое положение, размеры или видимость. В интерфейсе четыре метода:

public void componentMoved(ComponentEvent event) — вызывается, когда наблюдаемый компонент перемещается (в результате вызова команды setLocation(), работы менеджера размещения или еще по какой-то причине).

public void componentResized(ComponentEvent event) — вызывается, когда изменяются размеры наблюдаемого компонента.

public void componentHidden(ComponentEvent event) — вызывается, когда компонент становится невидимым.

public void componentShown(ComponentEvent event) — вызывается, когда компонент становится видимым.

Слушатель выбора элемента ItemListener

Слушатель ItemListener реагирует на изменение состояния одного из элементов, входящих в состав наблюдаемого компонента. Например, выпадающий список JComboBox состоит из множества элементов и слушатель реагирует, когда изменяется выбранный элемент. Также данный слушатель оповещается при выборе либо отмене выбора флажка JCheckBox или переключателя JRadioButton, изменении состояния кнопки JToggleButton и т.д. Слушатель обладает одним методом:

public void itemStateChanged(ItemEvent event).

Универсальный слушатель ActionListener

Среди многочисленных событий, на которые реагирует каждый элемент управления (и о которых он оповещает соответствующих слушателей, если они к нему присоединены), есть одно основное, вытекающее из самой сути компонента и обрабатываемое значительно чаще, чем другие. Например, для кнопки это щелчок на ней, а для выпадающего списка — выбор нового элемента.

Для отслеживания и обработки такого события может быть использован особый слушатель ActionListener, имеющий один метод:

public void actionPerformed(ActionEvent event).

У использования ActionListener есть небольшое преимущество в эффективности (так, при обработке нажатия на кнопку не надо реагировать на четыре лишних события — ведь даже если методы-обработчики пустые, на вызов этих методов все равно тратятся ресурсы). А кроме того очень удобно запомнить и постоянно использовать один класс с одним методам и обращаться к остальным лишь в тех относительно редких случаях, когда возникнет такая необходимость.

Обработка нажатия на кнопку ok легко переписывается для ActionListener:

```
ok.addMouseListener(new ActionListener() {
   public void actionPerformed(ActionEvent event) {
      if (loginField.getText().equals("Иван"))
         JOptionPane.showMessageDialog(null, "Вход выполнен");
      else JOptionPane.showMessageDialog(null, "Вход НЕ выполнен");
   }
});
```

ВАРИАНТЫ ЗАДАНИЙ

1. Реализуйте игру-угадайку, которая имеет одно текстовое поле и одну кнопку. Он предложит пользователю угадать число между 0-20 и дает ему три попытки. Если ему не удастся угадать, то будет выведено сообщение, что пользователь допустил ошибку в угадывнии и что число меньше / больше. Если пользователь попытался три раза угадать, то программу завершается с соответствующим сообщением. Если пользовательугадал, то программа должна тоже завершаться с соответствующим сообщением.
2. Реализация приложения Java, который имеет макет границы и надписи для каждой области в макете. Теперь определим события мыши, чтобы описать действия:

a. Когда мышь входит CENTER программа показывает диалоговое окно
(Добро пожаловать в)

b. Когда мышь входит WEST программа показывает диалоговое окно
(Добро пожаловать в Джидда)

c. Когда мышь входит SOUTH программа показывает диалоговое окно
(Добро пожаловать Абха)

d. Когда мышь входит в NORTH программа показывает диалоговое окно (Добро пожаловать в)

e. Когда мышь входит EAST программа показывает диалоговое окно
(Добро пожаловать в Дахране)

3. Реализация программу на Java с JTextArea и двумя меню:
Цвет: который имеет возможность выбора из три возможных : синий, красный и черный
Шрифт: тривида: “Times New Roman”, “MS Sans Serif”, “Courier New”.Вы должны написать прогамму, которая с помощью меню, может изменять шрифт и цвет текста, написанного в JTextArea

ЛАБОРАТОРНАЯ РАБОТА №7
КОЛЛЕКЦИИ, ОЧЕРЕДИ, СПИСКИ В JAVA

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Целью данной лабораторной работы является изучение работы с различными коллекциями в Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Коллекции.

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

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

Хотя в Java существует множество коллекций, но все они образуют стройную и логичную систему. Во-первых, в основе всех коллекций лежит применение того или иного интерфейса, который определяет базовый функционал. Среди этих интерфейсов можно выделить следующие:

- **Collection**: базовый интерфейс для всех коллекций и других интерфейсов коллекций
- **Queue**: наследует интерфейс Collection и представляет функционал для структур данных в виде очереди
- **Deque**: наследует интерфейс Queue и представляет функционал для двунаправленных очередей
- **List**: наследует интерфейс Collection и представляет функциональность простых списков
- **Set**: также расширяет интерфейс Collection и используется для хранения множеств уникальных объектов
- **SortedSet**: расширяет интерфейс Set для создания сортированных коллекций

- **NavigableSet**: расширяет интерфейс SortedSet для создания коллекций, в которых можно осуществлять поиск по соответствию
- **Map**: предназначен для созданий структур данных в виде словаря, где каждый элемент имеет определенный ключ и значение. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection

Эти интерфейсы частично реализуются абстрактными классами:

- **AbstractCollection**: базовый абстрактный класс для других коллекций, который применяет интерфейс Collection
- **AbstractList**: расширяет класс AbstractCollection и применяет интерфейс List, предназначен для создания коллекций в виде списков
- **AbstractSet**: расширяет класс AbstractCollection и применяет интерфейс Set для создания коллекций в виде множеств
- **AbstractQueue**: расширяет класс AbstractCollection и применяет интерфейс Queue, предназначен для создания коллекций в виде очередей и стеков
- **AbstractMap**: также расширяет класс AbstractCollection и применяет интерфейс Map, предназначен для создания наборов по типу словаря с объектами в виде пары "ключ-значение"
- **AbstractSequentialList**: также расширяет класс AbstractList и реализует интерфейс List. Используется для создания связанных списков

С помощью применения вышеописанных интерфейсов и абстрактных классов в Java реализуется широкая палитра классов коллекций - списки, множества, очереди, отображения и другие, среди которых можно выделить следующие:

- **ArrayList**: простой список объектов
- **LinkedList**: представляет связанный список
- **ArrayDeque**: класс двунаправленной очереди, в которой мы можем произвести вставку и удаление как в начале коллекции, так и в ее конце
- **HashSet**: набор объектов или хеш-множество, где каждый элемент имеет ключ - уникальный хеш-код
- **TreeSet**: набор отсортированных объектов в виде дерева
- **LinkedHashSet**: связанное хеш-множество

- **PriorityQueue**: очередь приоритетов
- **HashMap**: структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение
- **TreeMap**

Класс ArrayList.

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

ArrayList имеет следующие конструкторы:

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

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

Некоторые основные методы интерфейса List, которые часто используются в ArrayList:

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

- **E remove(int index)**: удаляет объект из списка по индексу index, возвращая при этом удаленный объект
- **E set(int index, E obj)**: присваивает значение объекта obj элементу, который находится по индексу index
- **void sort(Comparator<? super E> comp)**: сортирует список с помощью компаратора comp
- **List<E> subList(int start, int end)**: получает набор элементов, которые находятся в списке между индексами start и end

Используем класс ArrayList и некоторые его методы в программе:

```
import java.util.ArrayList;

public class CollectionApp {

    public static void main(String[] args) {

        ArrayList<String> states = new ArrayList<String>();
        // добавим в список ряд элементов
        states.add("Германия");
        states.add("Франция");
        states.add("Великобритания");
        states.add("Испания");
        states.add(1, "Италия"); // добавляем элемент по индексу 1

        System.out.println(states.get(1));// получаем 2-й объект
        states.set(1, "Дания"); // установка нового значения для 2-го объекта

        System.out.printf("В списке %d элементов \n", states.size());
        for(String state : states){

            System.out.println(state);
        }

        if(states.contains("Германия")){

            System.out.println("Список содержит государство Германия");
        }
```

```
        // удалим несколько объектов
        states.remove("Германия");
        states.remove(0);

        Object[] countries = states.toArray();
        for(Object country : countries){

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

Здесь объект ArrayList типизируется классом String, поэтому список будет хранить только строки. Поскольку класс ArrayList применяет интерфейс Collection<E>, то мы можем использовать методы данного интерфейса для управления объектами в списке.

Для добавления вызывается метод add. С его помощью мы можем добавлять объект в конец списка: states.add("Германия"). Также мы можем добавить объект на определенное место в списке, например, добавим объект на второе место (то есть по индексу 1, так как нумерация начинается с нуля): states.add(1, "Италия")

Метод size() позволяет узнать количество объектов в коллекции.

Проверку на наличие элемента в коллекции производится с помощью метода contains. А удаление с помощью метода remove. И так же, как и с добавлением, мы можем удалить либо конкретный элемент states.remove("Германия");, либо элемент по индексуstates.remove(0); - удаление первого элемента.

Получить определенный элемент по индексу мы можем с помощью метода get(): String state = states.get(1);, а установить элемент по индексу с помощью метода set: states.set(1, "Дания");

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

И поскольку класс ArrayList реализует интерфейс Iterable, то мы можем пробежаться по списку в цикле аля for-each: for(String state : states).

Хотя мы можем свободно добавлять в объект ArrayList дополнительные объекты, в отличие от массива, однако в реальности

ArrayList использует для хранения объектов опять же массив. По умолчанию данный массив предназначен для 10 объектов. Если в процессе программы добавляется гораздо больше, то создается новый массив, который может вместить в себя все количество. Подобные перераспределения памяти уменьшают производительность. Поэтому если мы точно знаем, что у нас список не будет содержать больше определенного количества элементов, например, 25, то мы можем сразу же явным образом установить это количество, либо в конструкторе: ArrayList<String> states = new ArrayList<String>(25);, либо с помощью метода ensureCapacity: states.ensureCapacity(25);

Класс LinkedList.

Обобщенный класс LinkedList<E> представляет структуру данных в виде связанного списка. Он наследуется от класса AbstractSequentialList и реализует интерфейсы List, Dequeue и Queue.

Класс LinkedList имеет следующие конструкторы:

- LinkedList(): создает пустой список
- LinkedList(Collection<? extends E> col): создает список, в который добавляет все элементы коллекции col

LinkedList содержит ряд методов для управления элементами, среди которых можно выделить следующие:

- **addFirst() / offerFirst()**: добавляет элемент в начало списка
- **addLast() / offerLast()**: добавляет элемент в конец списка
- **removeFirst() / pollFirst()**: удаляет первый элемент из начала списка
- **removeLast() / pollLast()**: удаляет последний элемент из конца списка
- **getFirst() / peekFirst()**: получает первый элемент
- **getLast() / peekLast()**: получает последний элемент

Рассмотрим применение связанного списка:

```
package collectionapp;

import java.util.LinkedList;

public class CollectionApp {

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

```
        LinkedList<String> states = new LinkedList<String>();

        // добавим в список ряд элементов
        states.add("Германия");
        states.add("Франция");
        states.addLast("Великобритания"); // добавляем на последнее
место
        states.addFirst("Испания"); // добавляем на первое место
        states.add(1, "Италия"); // добавляем элемент по индексу 1

        System.out.printf("В списке %d элементов \n", states.size());
        System.out.println(states.get(1));
        states.set(1, "Дания");
        for(String state : states){

            System.out.println(state);
        }
        // проверка на наличие элемента в списке
        if(states.contains("Германия")){

            System.out.println("Список содержит государство Германия");
        }

        states.remove("Германия");
        states.removeFirst(); // удаление первого элемента
        states.removeLast(); // удаление последнего элемента

        LinkedList<Person> people = new LinkedList<Person>();
        people.add(new Person("Mike"));
        people.addFirst(new Person("Tom"));
        people.addLast(new Person("Nick"));
        people.remove(1); // удаление второго элемента

        for(Person p : people){

            System.out.println(p.getName());
```

```
        }
        Person first = people.getFirst();
        System.out.println(first.getName()); // вывод первого элемента
    }
}
class Person{

    private String name;
    public Person(String value){

        name=value;
    }
    String getName(){return name;}
}
```

Здесь создаются и используются два списка: для строк и для объектов класса Person. При этом в дополнение к методам addFirst/removeLast и т.д., нам также доступны стандартные методы, определенные в интерфейсе Collection: add(), remove, contains, size и другие. Поэтому мы можем использовать разные методы для одного и того же действия. Например, добавление в самое начало списка можно сделать так: states.addFirst("Испания");, а можно сделать так: states.add(0, "Испания");

ВАРИАНТЫ ЗАДАНИЙ

1. Протестировать работу коллекции ArrayList.

2. Протестировать работу коллекции LinkedList.

3. Создать свою коллекцию, такую же как и ArrayList.

ЛАБОРАТОРНАЯ РАБОТА №8
РАБОТА С ФАЙЛАМИ

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Освоить на практике работу с файлами на языке Java. Получить практические навыки по чтению и записи данных в файл.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Запись файлов. Класс FileWriter

Класс FileWriter является производным от класса Writer. Он используется для записи текстовых файлов.

Чтобы создать объект FileWriter, можно использовать один из следующих конструкторов:

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(FileDescriptor fd)
FileWriter(String fileName)
FileWriter(String fileName, boolean append)
```

Так, в конструктор передается либо путь к файлу в виде строки, либо объект File, который ссылается на конкретный текстовый файл. Параметр append указывает, должны ли данные дозаписываться в конец файла (если параметр равен true), либо файл должен перезаписываться.

Запишем в файл какой-нибудь текст:

```
import java.io.*;

public class FilesApp {

    public static void main(String[] args) {

        try(FileWriter writer = new
FileWriter("C:\\SomeDir\\notes3.txt", false))
{
            // запись всей строки
Stringtext = "Мама мыла раму, раму мыла мама";
writer.write(text);
             // запись по символам
writer.append('\n');
```

```
            writer.append('E');

            writer.flush();
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
        }
    }
}
```

В конструкторе использовался параметр append со значением false - то есть файл будет перезаписываться. Затем с помощью методов, определенных в базовом классе Writer производится запись данных.

Чтение файлов. Класс FileReader

Класс FileReader наследуется от абстрактного класса Reader и предоставляет функциональность для чтения текстовых файлов.

Для создания объекта FileReader мы можем использовать один из его конструкторов:

```
FileReader(String fileName)
FileReader(File file)
FileReader(FileDescriptor fd)
```

А используя методы, определенные в базовом классе Reader, произвести чтение файла:

```
import java.io.*;

public class FilesApp {

    public static void main(String[] args) {

        try(FileReader reader = new
FileReader("C:\\SomeDir\\notes3.txt"))
        {
           // читаемпосимвольно
            int c;
            while((c=reader.read())!=-1){

                System.out.print((char)c);
            }
```

```
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
}
    }
}
```

ВАРИАНТЫ ЗАДАНИЙ

1. Реализовать запись в файл введённой с клавиатуры информации
2. Реализовать вывод информации из файла на экран
3. Заменить информацию в файле на информацию, введённую с клавиатуры
4. Добавить в конец исходного файла текст, введённый с клавиатуры

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Роберт Лафоре, Структуры данных и алгоритмы в Java
2 В. Белов, В. Чистякова Алгоритмы и структуры данных. УчебникИнфра-м 2016 240 стр. ISBN: 5906818256
3 Богачев К.Ю. Основы параллельного программирования– М.: Бином. Лаборатория знаний, 2003. -.342 с.
4 Зыль С.Н. Операционная система реального времени QNX Neutrino: от теории кпрактике – Изд. 2-е - СПб.: БХВ-Петербург, 2004. – 192 с.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Зорина Н.В. Курс лекций по Объектно-ориентированному программированию на Java, МИРЭА, Москва, 2016

2. Программирование на языке Java: работа со строками и массивами. Методические указания. [Электронный ресурс] : Учебно-методические пособия — Электрон. дан. — СПб. : ПГУПС, 2015. — 24 с.
3. Кожомбердиева, Г.И. Программирование на языке Java: создание графического интерфейса пользователя: учеб. пособие. [Электронный ресурс] : Учебные пособия / Г.И. Кожомбердиева, М.И. Гарина. — Электрон. дан. — СПб.: ПГУПС, 2012. — 67 с.
4. Вишневская, Т.И. Технология программирования. Часть 1. [Электронный ресурс] / Т.И. Вишневская, Т.Н. Романова. — Электрон. дан. — М. : МГТУ им. Н.Э. Баумана, 2007. — 59 с.