СОДЕРЖАНИЕ

Практическая работа № 1. Знакомство со средой разработки. Синтаксис и основные управляющие конструкции языка Джава

Цель: введение в разработку программ на языке программирования Джава.

Теоретические сведения

Язык Джава— это объектно-ориентированный язык программирования, с со строгой инкапсуляцией и типизацией. Программы, написанные на языке, Джава могут выполняться под управлением различных операционныхсистемах при наличии необходимого ПО – Java Runtime Environment.

Для того чтобы создать и запускать программы на языке Джава необходимо следующее ПО:

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

Установка необходимого для разработки на Джава программного обеспечения

Для того чтобы скачать среду разработки, можно воспользоваться следующими ссылками:

1. Программа IntelliJ IDEA это профессиональная интегрированная среда разработки для разработки программ на Джава: https://www.jetbrains.com/idea/download/#section=windows
2. Программа “NetBeans IDE”: https://netbeans.org/downloads/
3. JDK: http://www.oracle.com/technetwork/java/javase/downloads/index.html

По умолчанию, скаченный JDK установится в папку с таким адресом: C:\Program Files\Java на компьютере с ОС Windows

Начало работы с программой IntelliJ IDEA

После установки одной из сред разработки («IntelliJ IDEA» или «NetBeans IDE») можно начать создавать проекты. Далее будет показано, как начать новый проект на примере программы «IntelliJ IDEA». Рассмотрим подробно, по шагам процесс создания:

1. В открытом окне программы выбираем «Create New Project».

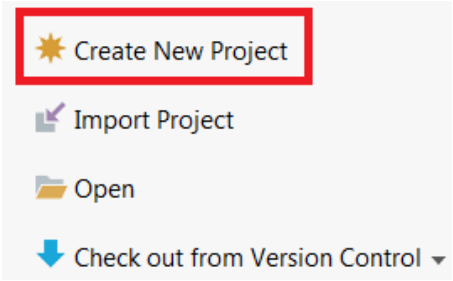

Рисунок 1.1. Пункт в меню для выбора для создания проекта

2. Щёлкаем «New», чтобы загрузить JDK.

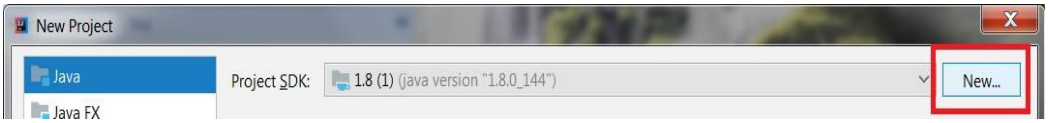

Рисунок 1.2. Поле – путь к SDK(скачали ранее)

В этом поле настраивается путь к SDK, котрый вы предварительно скачали с сайта oracle.

3. Из выпадающего списка папок выбираем «Program Files».

Рисунок 1.3. Пункт выбора папки с SDK

4. В «Program Files» выбираем папку «Java».

Рисунок 1.4. Папка с SDK

5. Далее выбираем папку «jdk…».

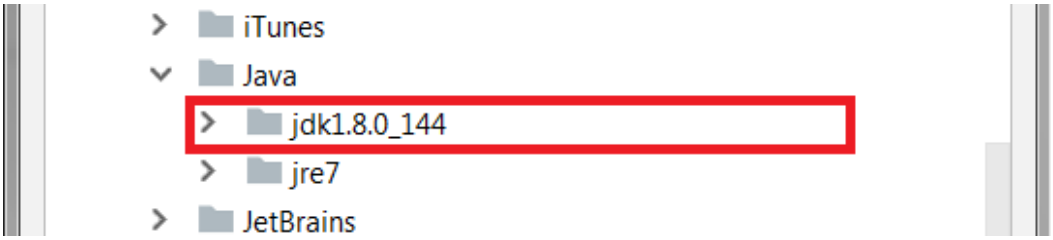

Рисунок 1.5. Выбираем папку с jdk

6. Затем дважды нажимаем «Next» в нижнем правом углу.

Рисунок 1.6. Подтверждение выбора

Выбираем название для будущего проекта и затем нажимаем кнопку «Finish».

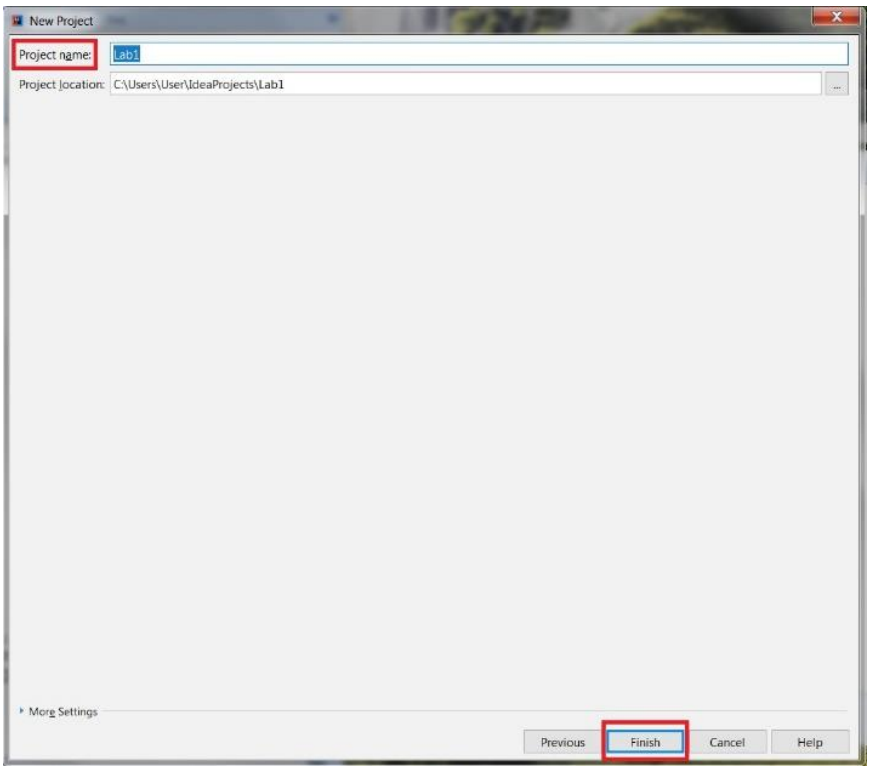

Рисунок 1.7. Поле для ввода названия проекта

7. Щёлкаем правой кнопкой мыши по папке «src» в дереве файлов проекта и создаем новый пакет.

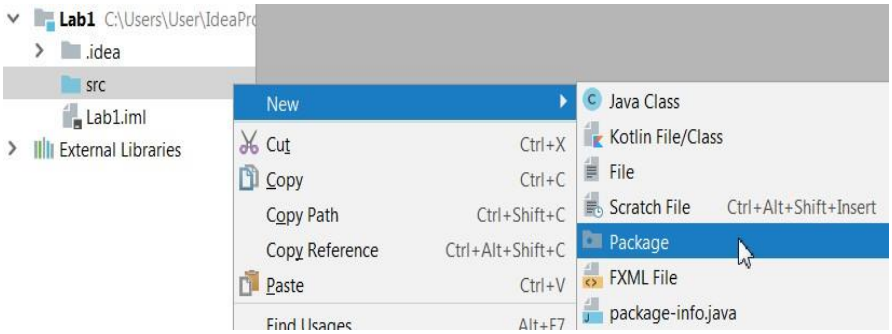

Рисунок 1.8. Папка с исходным кодом

8. Вводим название пакета. «Package» – это оператор, который сообщает транслятору, в каком пакете должны определяться содержащиеся в данномфайле классы.

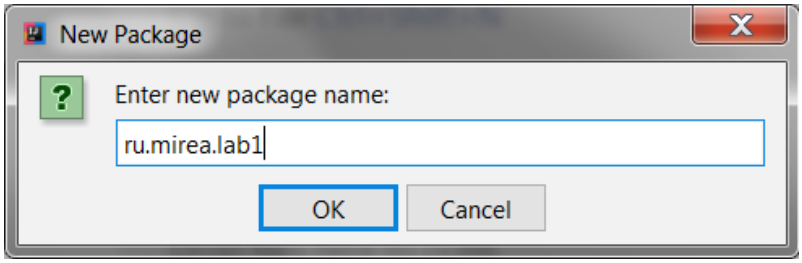

Рисунок 1.9. Поле для ввода названия пакета

Щелкаем по созданному пакету правой кнопкой мыши и создаем новый класс.

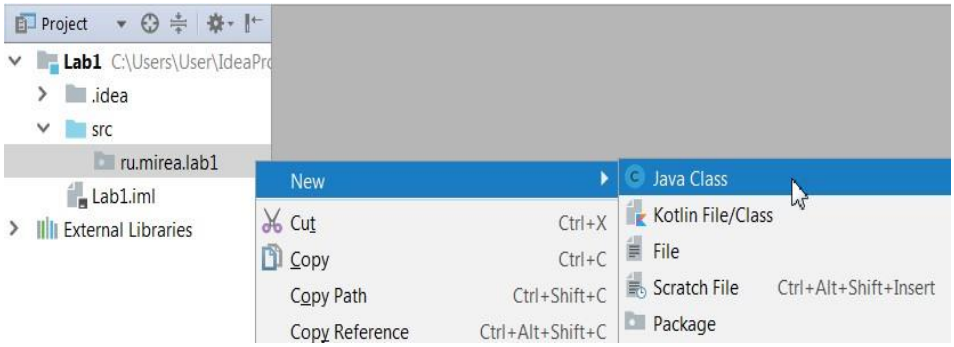

Рисунок 1.10. Меню выбора для создания класса

9. Новый проект создан. Теперь можно приступать к написанию кода.

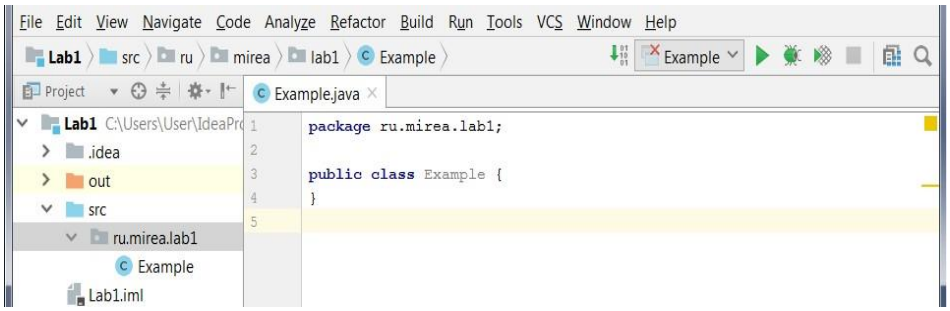

Рисунок 1.11. Окно проекта и файл класса

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 Джава проект, после чего необходимо создать пакет и в нем создать какой-либо класс. По правилам именования пакетов используются только строчные буквы, в названии класса первая буква имени должна быть заглавная. 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.

Листинг 1.1 Пример объявления класса:

```
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 Джава программы, выполнение начнется с метода main().

Переменные класса.

Чтобы объявить переменную, необходимо указать тип переменной и ее имя. Тип переменной может быть разным: 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 целыечисла, а числам с плавающей запятой - дробные. Целые числа обозначаются цифрами от 0 до 9, а дробные можно записывать отделяюцелую 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 языке Джава, так как ему можно присваивать значение, не создавая экземпляра класса (Джава это сделает автоматически). Этот класс предназначен для представления строк. Строковое значение записывается буквами внутри двойных кавычек.

Листинг 1.2 Пример присваивания значений переменным

```
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, остатокот деления. Эти операции обозначаются соответственно “+”, “-”, “*”, “/”, “%”. Для чисел с плавающей запятой применимы операции сложения, вычитания, умножения, деления. Для строк применима операция “+”, обозначающая конкатенацию, или слияние строк.

Массивы в Джава

Массив — это конечная последовательность 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 нашего прримера объявления массива.

Для того чтобы получить какой-либо элемент массива, нужно после имени массива в квадратных скобках индекс, или порядковый номер элемента. Нумеруются в массивах начинается с нуля, как и в языке С.С++. Например, чтобы получить 5 элемент массива, можно написать так: b[4].

Условные конструкции в Джава

Условие — это конструкция, позволяющая выполнять то или другое действие, в зависимости от логического значения, указанного в условии. Синтаксис создания условия на Джава 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.

Циклические конструкции в Джава

Цикл — это конструкция, позволяющая выполнять определенную часть кода несколько раз. В Джава 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.

Итерационный цикл for

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 которомбудет выполняться тело цикла (пока i<10), после второй точки с запятой указывается как будет изменяться переменная i (увеличиваться на единицу каждый раз с помощью операции инкремента “++”). Прописыватьусловие, объявлять переменную и указывать изменение переменной вцикле for не обязательно, но обязательно должны быть точки с запятой.

Цикл с предусловием while

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 с постусловием do while

Цикл вида 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 пакета java.util.

Этот класс надо импортировать в те программе, где он будет использоваться. Это делается до начала определения 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 Джава представлен объектом — System.in. А

стандартный поток вывода (дисплей) — уже знакомым вам объектом System.out. Есть ещё стандартный поток для вывода ошибок —System.err

Листинг 1.3 – Пример чтения данных ввода с клавиаьуры

```
import java.util.Scanner; // импортируем  класс
import java.util.Scanner; // импортируем  класс
public class Main {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in); /
/ создаёмобъект класса  Scanner
int i = 2; System.out.print("Введитецелоечисло: ");
if(sc.hasNextInt()) {
/* hasNextInt()возвращает истинну если с потока ввода
можно считать целое число*/
i = sc.nextInt(); /* считывает целое число из потока
ввода  и  сохраняет  в переменную i*/
System.out.println(i*2);
} else {
System.out.println("Вы ввели не целое число");
}}
```

В Джава 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 поочерёдно записывается ввод пользователя, а далее на экран выводится одна строка, полученная объединением введённых последовательностей символов.

Листинг 1.4 – Пример чтения строк и их конкатенации

```
import java.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 впотоке 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). Некотрые из метолов:

- 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 варгументе 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 номерсимвола, но нумероваться символы начинают с нуля). char charAt(n) — возвращает код символа, находящегося в строке под индексом n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).

Листинг 1.5 – Пример работы с методами String

```
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);
i = s1.indexOf('f');
System.out.println(i);
i= s1.indexOf('r');
System.out.println(i);
i =s1.lastIndexOf('f');
System.out.println(i);
i = s1.indexOf('t');
System.out.println(i);
i = s1.indexOf('r',3);
System.out.println(i);
}}
```

Пример программы, которая выведет на экран индексы всех пробелов в строке, введённое пользователем с клавиатуры:

Листинг 1.6 – Пример работы со строками

```
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 методключевым словом static.

Нужно обязательно указать тип возвращаемогозначения 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 определения метода класса:

```
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 создаваяэкземпляра 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);
```

Задания на практическую работу № 1

1. Создать проект в IntelliJ IDEA
2. Создать свой собственный Git репозитарий
3. Написать программу, в результате которой массив чисел создается с помощью инициализации (как в Си) вводится и считается в цикле сумма элементов целочисленного массива, а также среднее арифметическое его элементов результат выводится на экран. Использовать цикл for.
4. Написать программу, в результате которой массив чисел вводится пользователем с клавиатуры считается сумма элементов целочисленного массива с помощью циклов do while, while, также необходимо найти максимальный и минимальный элемент в массиве, результат выводится на экран.
5. Написать программу, в результате которой выводятся на экран аргументы командной строки в цикле for.
6. Написать программу, в результате работы которой выводятся на экран первые 10 чисел гармонического ряда (форматировать вывод).
7. Написать программу, которая с помощью метода класса, вычисляет факториал числа (использовать управляющую конструкцию цикла), проверить работу метода.
8. Результаты выполнения практической работы залить через IDE в свой репозитарий и продемонстрировать преподавателю.

Практическая работа № 2. Объектно-ориентированное программирование в Джава. Классы в Джава

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

Теоретические сведения

Для начала разберем, что такое модификаторы доступа в Java. В Java существуют следующие модификаторы доступа:

- private: данные класса доступны только внутри класса;
- protected: данные класса доступны внутри пакета и в наследниках;
- public: данные класса доступны всем.

В качестве примера рассмотрим программу, описывающую окружность. Создадим класс Circle. Каждая окружность имеет несколько параметров:

- Координаты по оси x;
- Координаты по оси y;
- Радиус r;
- Цвет colour.

Объявим переменные класса:

```
private double x;
private double y;
private double r;
private String colour;
```

Как мы уже знаем, тип данных double приемняется для числовых значений, а String для текстовых.

Одним из стандартных подходов при проектировании классов на языке Джава является управление доступом к атрибутам класса через пару методов get и set. Метод get позволяет получить значение поля, set — установить новое значение. Общий принцип именования этих методов (называемых, также, геттером и сеттером). Для того, чтобы автоматитчески добавить эти методы в класс, щелкаем правой кнопкой мыши (или путем сочетания клавиш Alt+Insert) и из выпадающего меню выбираем «Generate»[1], а затем щелкаем по «Getter» или «Setter», в зависимости от того, что нам необходимо.

Листинг 2.1 – Пример сеттеров и геттеров для класса

```
public double getX()
```

[1] Корректно для IntelliJ IDEA

```
{return x;}
public void setX(double x)
{this.x = x;}
public double getY()
{return y;}
public void setY(double y)
{this.y = y;}
public double getR()
{return r;}
public void setR(double r)
{this.r = r;}
public String getColour()
{return colour;}
public void setColour(String colour)
{this.colour = colour;}
```

Обратите внимание, что в теле метода автоматически прописались два вида строчек: return (для getter) и this. (для setter). Return пишется для возврата значений переменной (чтения), this. пишется в качестве ссылки на данные переменной.

Затем также, как мы вызывали методы getter и setter, вызываем метод конструктор класса (Constructor). Конмструктор — 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 класса нужен для автоматической инициализации переменных.

Листинг 2.2 – Пример конструктора для класса Circle

```
public Circle(double x, double y, double r, String colour)
{
this.x = x;
this.y = y;
this.r = r;
this.colour = colour;
}
```

Далее мы также вызываем еще один метод – toString().Метод toString в Java используется для предоставления ясной и достаточной информации об объекте (Object) в удобном для человека виде.

Листинг 2.3 – Пример переопределения метода toString()

```
@Override
    public String toString() {
        return "Circle{" +
"x=" + x +", y=" + y +
", r=" + r +
", colour='" + colour + '\'' +
 '}';
 }
```

Последним этапом формирования класса Circle будет создание метода getLength(). В теле данного метода мы объявляем переменную *c*, которая обозначает длину нашей окружности. Длина окружности описывается формулой: c = 2*PI*r. В Java нельзя просто так написать PI, поэтому следует обратиться к библиотеке Math. В конце тела метода необходимо возвратить значение *c*. Получается такая запись:

Листинг 2.3 – Пример функции класса getLength()

```
public double getLength(){
        double c;
        c = 2*Math.PI*r;
        return c;
    }
}
```

Последняя фигурная скобка не является лишней – она закрывает тело нашего класса Circle.

Следующим этапом написания нашей программы будет создание в этом же пакете, в котором мы работали, класса Tester (часто называют Main). Затем добавляем необходимые библиотеки: lang и Scanner (в дальнейшем мы будем использовать ввод с клавиатуры):

```
import java.lang.*;
import java.util.Scanner;
```

Далее прописываем аббревиатуру “psvm” для создания статического метода main(String[]). Следующим шагом будет объявление переменной *r*. Второй раз мы объявляем переменную с таким же именем, но в другом классе, чтобы ввести значение *r* с клавиатуры и посмотреть, как меняется значение *c*:

```
public class Tester {
    public static void main(String[] args) {
            double r;
```

Затем мы создаем экземпляр (k1) класса Circle:

```
    Circle k1 = new Circle(x:3.1, y:4.1, r:5.1,
colour:“red”);
```

Далее прописываем стандартную процедуру вывода значений:

```
System.out.println("Длина окружности = " +
 k1.getLength() + "см\n");
```

После ввода этой строчки на консоли выведется длина окружности *c*, при радиусе *к*, который мы ввели для экземпляра класса (r:5.1).

Включаем ввод с клавиатуры и пишем процедуру «Вывод», в которой будет содержаться информация о том, что нам необходимо ввести, и пишем переменную, которую нам необходимо будет ввести. Делается это так: r = source.nextDouble();. В итоге получается следующее:

```
Scanner source = new Scanner(System.in);
System.out.println("Введите радиус ");
    r = source.nextDouble();
```

Пишем k1.setR(r);. Это необходимо для того, чтобы при работе программа обращалась уже к новому значению *r*, а не к тому, которое мы ввели для экземпляра класса. Завершением класса Tester будет написание процедуры “Вывод”.

```
k1.setR(r);
    System.out.println("\nДлина окружности = " +
k1.getLength() + "см");
    }
}
```

В итоге получаем следующий код программы, включающий в себя 2 класса. На рис. 2.1 представлен класс Circle

Circle.java

```java
package ru.mirea.it.java;
public class Circle {
    private double x;
    private double y;
    private double r;
    private String colour;

    public Circle(double x, double y, double r, String colour) {...}

    public double getX() { return x; }

    public void setX(double x) { this.x = x; }

    public double getY() { return y; }

    public void setY(double y) { this.y = y; }

    public double getR() { return r; }

    public void setR(double r) { this.r = r; }

    public String getColour() { return colour; }

    public void setColour(String colour) { this.colour = colour; }
    public double getPerimetr() { return Math.PI*this.getR(); }
    @Override
    public String toString() {
        return "Circle{" +
                "x=" + x +
                ", y=" + y +
                ", r=" + r +
                ", colour='" + colour + '\'' +
                '}';
    }

```

Рисунок 2.1. Класс Circle.java

На рис. 2.2 представлен класс Tester в котором создается объект класса Circle и вызываются его методы.

Tester.java | Circle.java

```
1  package ru.mirea.lab7;
2  import java.lang.*;
3  import java.util.Scanner;
4
5  public class Tester {
6      public static void main(String[] args) {
7          double r;
8          Circle k1 = new Circle( x: 3.1, y: 4.1, r: 5.1,  colour: "\u001B[31m");
9          System.out.println("Длина окружности = " + k1.getPerimetr() + "см");
10         Scanner source = new Scanner(System.in);
11         System.out.println("Введите радиус ");
12         r = source.nextDouble();
13         k1.setR(r);
14         System.out.println("\nДлина окружности = " + k1.getPerimetr() + "см");
15     }
16 }
17
```

Рисунок 2.2. Класс Tester.java

На рисунке 2.3 приведен пример результат работы программы.

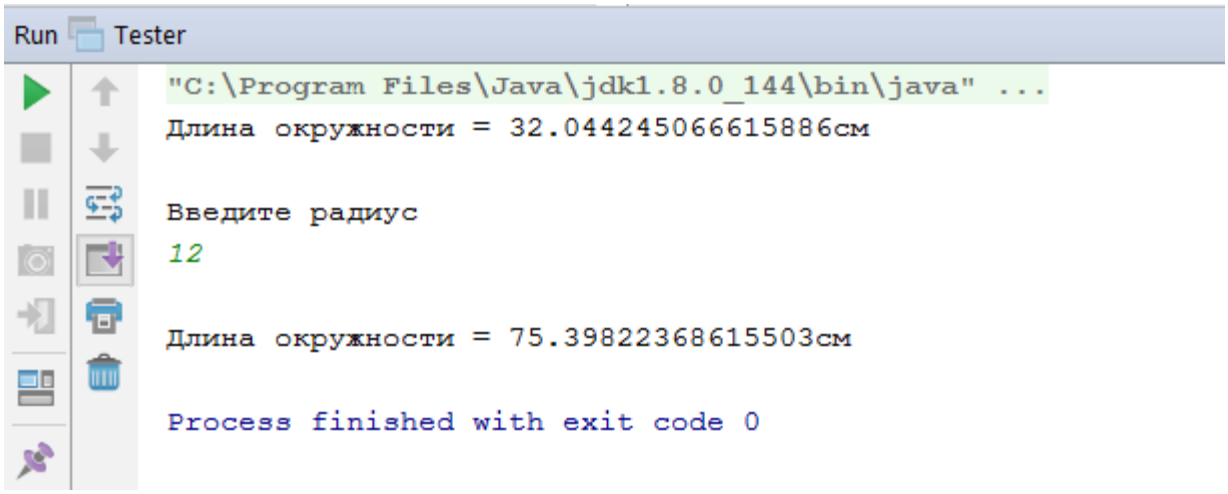

Рисунок 2.3. Результат работыTester.java

Задания на практическую работу № 2

1. По диаграмме класса UML описывающей сущность Автор. Необходимо написать программу, которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме класса на рисунке 2.4.

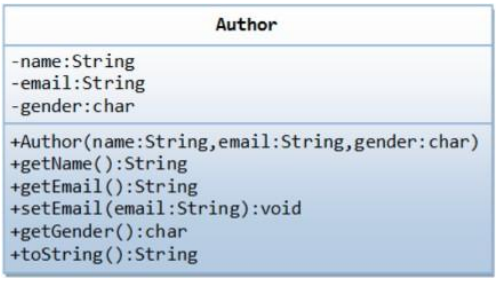

Рисунок 2.4. Диаграмма класса Author.

2. По UML диаграмме класса, представленной на рис. 2.5 написать программу, которая состоит из двух классов. Один из них Ball должен реализовывать сущность мяч, а другой с названием TestBall тестировать работу созданного класса. Класс Ball должен содержать реализацию методов, представленных на UML. Диаграмма на рисунке описывает сущность Мяч написать программу. Класс Ball моделирует движущийся мяч.

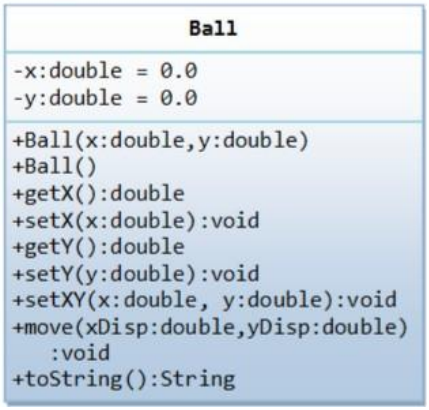

Рисунок 2.5. Диаграмма класса Ball.

3. Создать класс точка Point, описывающий точку на плоскости. Создать Circle класс, в котором одно поле представляет точку – центр окружности, и добавить другие свойства, позволяющие задать точку на плоскости. Создать третий класс Tester который использует для хранения объектов массив объектов Circle и второе поле количество элементов в массиве.

4. Разработайте класс Shop для, реализуйте методы добавления и удаления компьютеров в магазине, добавьте метод поиска в магазине компьютера, нужного пользователю. Протестируйте работу созданных классов. Данные для заполнения массива компьютеров вводятся с клавиатуры пользователем. Для этого реализуйте интерфейс.

5. Разработайте и реализуйте класс Dog (Собака), поля класса описывают кличку и возраст собаки. Необходимо выполнить следующие действия: определить конструктор собаки, чтобы принять и инициализировать данные экземпляра., включить стандартные методы (аксессоры) для получения и установки для имени и возраста, включить метод для перевода возраста собаки в “человеческий” возраст (возраст семь раз собаки), включите метод ToString, который возвращает описание экземпляра собаки в виде строки. Создание класса тестера под названием ПитомникСобак, реализует массив собак и основной метод этого класса позволяет добавить в него несколько объектов собаки.

6. Создать класс, описывающий модель окружности (Circle). В классе должны быть описаны нужные свойства окружности и методы для получения и изменения этих свойств. Добавить методы для расчета площади круга и длины

окружности, а также метод позволяющий сравнивать две окружности. При помощи класса CircleTest, содержащего статический метод main(String[] args), протестировать работу класса Circle.

7. 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). Создать класс книжная полка, в котором поля данных класса это массив объектов типа книги (Book, и количество книг на книжной полке. Написать методы класса, которые возвращают книги с самым поздним и самым ранним сроком издания. Написать метод класса, позволяющий расставить книги на книжной полке в порядке возрастания года выпуска. Используйте реализацию отношений композиция классов[2]

8. Напишите программу, которая меняет местами элементы одномерного массива из String в обратном порядке. Не используйте дополнительный массив для хранения результатов.

9. Напишите программу Poker.java, которая должна имитировать раздачу карт для игры в покер. Программа получает число n, задаваемое с консоли пользователем, и раздает карты на n игроков (по 5 карт каждому) из перетасованной колоды. Разделяйте пять карт, выданных каждому игроку, пустой строкой.

10. Напишите программу HowMany.java, которая определит, сколько слов Вы ввели с консоли

[2] Композиция https://ru.wikipedia.org/wiki/Диаграмма_классов

Практическая работа № 3. Классы Math и Random. Классы оболочки

Цель данной практической работы - изучить работу с классами Math и Random основные концепции объектно-ориентированного программирования, научиться программировать математические вычисления с использованием этих классов, а также познакомиться с классами оболочками и их использованием в Джава программах и научиться форматировать вывод строк.

Теоретические сведения

Класс Math

Класс Java Math предоставляет ряд методов для работы выполнения математических вычислений, например таких как как min (), max (), sqrt(), pow(), sin(), cos(), tan(), round (), abs ( ) так далее. В классе также ест константа число PI. Все методы класса публичные и статические, поэтому вы можете вызывать их, обратившись напрямую через точку к классу, не создавая объект типа класс. Если размер равен int или long и результаты выходят за пределы диапазона значений, методы addExact (), subtractExact (), multiplyExact () и toIntExact () вызывают исключение ArithmeticException.

Для других арифметических операций, таких как увеличение, уменьшение, деление, абсолютное значение и отрицание, переполнение происходит только с определенным минимальным или максимальным значением. При необходимости его следует сравнивать с максимальным и минимальным значением. Класс Java Math имеет множество методов, которые позволяют решать на Джава математические задачи.

```
public class JavaMathExample1{
public static void main(String[] args){
double x = 28;
double y = 4;
// вернет maximum из двух чисел
System.out.println("Maximum number of x and y is: "
+Math.max(x, y));
// вернет квадратный корень из y
System.out.println("Square root of y is: " +
Math.sqrt(y));
//вергнет 28 в четвертой степени 28*28*28*28
System.out.println("Power of x and y is: " +
Math.pow(x, y));
// вернет логарифм заданного значения
```

```
    System.out.println("Logarithm of x is: " +
Math.log(x));
    System.out.println("Logarithm of y is: " +
Math.log(y));
    // вернет логарифм из десяти по основанию 10
    System.out.println("log10 of x is: " + Math.log10(x));
    System.out.println("log10 of y is: " + Math.log10(y));

    // вернет log из x + 1
    System.out.println("log1p of x is: " +Math.log1p(x));
    /* Метод Math.exp() — возвращает натуральный логарифм
по основанию e и аргументу — показателю степени, где e —
иррациональная константа, равная приблизительно 2,71828182
*/
    System.out.println("exp of a is: " +Math.exp(x));
    System.out.println("expm1 of a is: " +Math.expm1(x));
        }
    }
```

Генерация случайных чисел в Джава

Язык Джава предоставляет три способа генерации случайных чисел с использованием некоторых встроенных методов и классов, перечисленных ниже:

1. Класс Random
2. Метод random ()
3. класс ThreadLocalRandom

Класс Random находится в пакете java.util, соответственно для работы с ним вы должны импортировать этот пакет. Метод random() это метод класса Math, он может генерировать случайные числа типа double. Рассмотрим первый подход, с помощью класса Random. Этот класс имеет в своём составе различные методы для создания случайных чисел. Все они публичные и статические. Чтобы использовать этот класс для генерации случайных чисел, мы должны сначала создать экземпляр этого класса, а затем уже вызвать его методы, например такие как nextInt (), nextDouble (), nextLong () и тому подобные. С помощью этого класса мы можем генерировать случайные числа различных типов: integer, float, double, long, boolean. Мы можем передать аргументы методам для определения верхней границы диапазона генерируемых чисел. Например, если вызвать метод

nextInt (6) с аргументом 6, то будет генерироваться число в диапазоне от 0 до 5 включительно.

Листинг 3.1 – Пример генерации случайного числа

```
import java.util.Random;
public class generateRandom{
public static void main(String args[]){
// Создаем экземпляр класса Random
Random rand = new Random();

// Генерируем рандомно целые числа в диапазоне 0 to 999
int rand_int1 = rand.nextInt(1000);
int rand_int2 = rand.nextInt(1000);

// Печатаем полученные числа тип - int
System.out.println("Random Integers: "+rand_int1);
System.out.println("Random Integers: "+rand_int2);

// Генерируем с помощью Random тип double
double rand_dub1 = rand.nextDouble();
double rand_dub2 = rand.nextDouble();

// Печатаем полученные числа - тип double
System.out.println("Random Doubles: "+rand_dub1);
System.out.println("Random Doubles: "+rand_dub2);
    }
}
```

Результат работы программы на листинге 3.1 представлен ниже:

```
Random Integers: 547

Random Integers: 126

Random Doubles: 0.8369779739988428

Random Doubles: 0.5497554388209912
```

Рассмотрим второй способ с помощью метода Math.random (). Класс Math содержит различные методы для выполнения различных числовых операций, таких как вычисление возведения в степень, логарифмы и т. Д. Один из этих методов - random(), возвращает значение типа double с положительным знаком, больше или равно 0,0 и меньше 1,0. Возвращаемые значения выбираются

псевдослучайно. Этот метод может генерировать только случайные числа типа Double. Программа на листинге 3.2 демонстрирует как использовать этот метод при написании программ:

Листинг 3.2 – Пример использования метода random()

```
import java.util.*;
public class generateRandom{
public static void main(String args[]){
// Сгенерируем рандомно double
System.out.println("Random doubles: " +
Math.random());
System.out.println("Random doubles: " +
Math.random());
    }
}
```

Результат работы программы на листинге 3.2 представлен ниже:

```
Random doubles: 0.13077348615666562
Random doubles: 0.09247016928442775
```

Теперь рассмотрим третий подход для генерации случайного числа, с помощью класса ThreadLocalRandom. Этот класс представлен в java 1.7 для генерации случайных чисел типа целых, двойных, логических и т. Д. В программе ниже объясняется, как использовать этот класс для генерации случайных чисел:

Листинг 3.3 – Пример использования класса ThreadLocalRandom.

```
// сгенерируем рандомно числа
import java.util.concurrent.ThreadLocalRandom;
public class generateRandom{
public static void main(String args[]){
// создаем случ. целые числа в диапазоне 0 to 999
int rand_int1 =
ThreadLocalRandom.current().nextInt();
int rand_int2 =
ThreadLocalRandom.current().nextInt();
// печатаем случайные целые числа
System.out.println("Random Integers: " + rand_int1);
System.out.println("Random Integers: " + rand_int2);

// Генерируем рандомно double
```

```
    double rand_dub1 =
ThreadLocalRandom.current().nextDouble();
    double rand_dub2 =
ThreadLocalRandom.current().nextDouble();
    // печатаем случайные целые числа double
    System.out.println("Random Doubles: " + rand_dub1);
    System.out.println("Random Doubles: " + rand_dub2);

    // Генерируем рандомно boolean
    boolean rand_bool1 =
ThreadLocalRandom.current().nextBoolean();
    boolean rand_bool2 =
ThreadLocalRandom.current().nextBoolean();

    // печатаем случайные числа Boolean
    System.out.println("Random Booleans: " + rand_bool1);
    System.out.println("Random Booleans: " + rand_bool2);
        }
    }
```

Результат работы программы на листинге 3.3 представлен ниже:

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of a is: 1.4462570642914 75E12
expm1 of a is: 1.4462570642904 75E12
```

Для того, чтобы более подробно узнать про генерацию случайных чисел обратитесь к документации по языку Джава на сайте компании Oracle.[3]

Классы оболочки

В языке Джава у каждого примитивного типа есть соответствующий этому типу ссылочный или объектный тип класс – оболочка или обертка.

[3] https://docs.oracle.com

Ссылочный тип данных — это объект. В табл. 3.1 приведены примитивные типы данных и соответствующие им классы обертки:

Таблица 3.1 Соответствие примитивных и ссылочных типов

Примитивный тип	**Тип оболочка**
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Типы оболочки используются в Джава, поскольку контейнеры не могут хранить примитивные типы данных. Поэтому необходимо выполнять преобразование примитивного типа к ссылочному. Это называется автоупаковка. Обратное преобразование называется автораспаковка.

Рассмотрим на примере класса Integer работу методов класса оболочки. Класс Integer — это класс оболочка для примитивного типа int. У класса Integer есть одно только поле типа int. Так как Integer это оболочка, то основгная его задача предоставить различные методы для работы с int, а также ряд методов для преобразования int в String и String в int. Пример работы с классом:

```
Integer num = 30;
int i = 15;
Integer a = i; //это автоупаковка
System.out.println(a); // 15
Integer x = new Integer(10); //конструктор Integer
System.out.println(x);
int b = x;//автораспаковка
```

Методы класса Integer

У класса Integer есть метод Integer.parseInt(String s). метод возвращает целое число, а именно возвращают примитивный тип данных int. Также у него есть переопределений метод Integer.toString() который возвращает строку (String) то есть символьное представление числа.

```
System.out.println(Integer.parseInt("0011", 2)); // 3
System.out.println(Integer.parseInt("10", 8)); // 8
System.out.println(Integer.parseInt("F", 16)); // 15
```

У данных методов есть аналог — метод valueOf().Отличие от parseInt в том, что результатом работы valueOf () будет объект Integer, а не int.

Форматирование вывода

Язык Джава предоставляет возможность использовать форматирование вывод с помощью использования форматно строки функции `printf` () функции. Для задания спецификаторов вывода используются те же самые управляющие символы, что и в языке Си. Пример использования спецификаторов представлен на листинге 3.4

Листинг 3.4 – Пример использования функции printf ()для форматирования вывода

```
public class Test{
public static void main(String args[]){
double x = 11.635;
System.out.printf("Значение e = %.3f%n", Math.E);
System.out.printf("exp(%.3f) = %.3f%n", x,
Math.exp(x));
    }
}
```

Спецификаторы формата вывода

Рассмотрим общий синтаксис спецификаторов выводы для его форматирования для общих типов, символьных и числовых:

`%[индекс_аргумента$][флаги][ширина][.точность]спецификатор`

Спецификаторы индекс аргумента, флаг, ширина и точность не являются обязательными.

- Индекс аргумента— это целое число i, указывающее, что здесь следует использовать i-й аргумент из списка аргументов
- Флаги– это набор символов, используемых для изменения формата вывода
- Ширина поля вывода — это положительное целое число, которое указывает минимальное количество символов, которое должно быть записано на выходе
- Точность — это целое число, обычно используемое для ограничения количества символов, конкретное поведение которых зависит от преобразования.
- спецификатор является обязательной частью. Это символ, указывающий, как должен быть отформатирован аргумент,

форматирование зависит от типа выводимого значения. Набор допустимых преобразований для данного аргумента зависит от типа данных аргумента.

Спецификатор символа - %c, спецификатор целого числа %d, %i, спецификатор вещественного числа %lf, f%, спецификатор строки %s.

В нашем примере выше, если мы хотим указать номер аргумента явно, мы можем записать его, используя индексы аргументов 1 $ и 2 $. Оба они являются первым и вторым аргументом соответственно:

```
String greetings = String.format( "Hello %2$s, welcome to %1$s !", "MIREA", "students");
```

Вы можете использовать спецификаторы вывода без аргументов, тогда следует использовать такой шаблон формы записи:

```
%[флаги][ширина]спецификатор
```

Форматирование вывода. Класс Formatter

В языке Джава для форматирования строкового вывода часто используется класс Formatter. Для его использования нужно подключить пакет java.util. Этот класс обеспечивает преобразование формата вывода (спецификаторы формата) позволяющие выводить числа, строки, время и даты практически в любом понравившемся вам формате. В классе Formatter есть метод format(), который преобразует переданные в него параметры в строку заданного формата и сохраняет в объекте типа Formatter. Рассмотрим использование класса Formatter. Для начала необходимо создать объект класса Formatter, а потом уже вызывать метод format() для форматирования вывода.

Листинг 3.5 – Пример использования Formatter

```
import java.util.Formatter;
public class public class FormatDemo1 {
public static void main(String[] args) {
Formatter fmt = new Formatter();
fmt.format("This %s is about %s %c ", "course", "java", '8');
//вместо первого %s подставится строка course
//вместо второго %s подставится строка java
//вместо %c подставится символ 8
System.out.print(f);
}}
```

Рассмотрим еще один пример, на листинге 3.6 приведен код, демонстрирующий работу с форматированием вывода вещественных чисел

Листинг 3.6 – Пример использования Formatter

```
import java.util.Formatter;
public class FormatDemo2 {
public static void main(String[] args) {
double x = 1000.0 / 3.0;
System.out.println("Строка без форматирования: " +
x);
Formatter formatter = new Formatter();
formatter.format("Строка с форматированием: %.2f%n",
x);
formatter.format("Строка с форматированием: %8.2f%n",
x);
formatter.format("Строка с форматированием:
%16.2f%n", x);
System.out.println(formatter);
    }
}
```

Форматирование чисел. Классы NumberFormat и DecimalFormat

Классы NumberFormat и DecimalFormat находятся в пакете java.text. В языке Джава класс NumberFormat языка Java используется для форматирования чисел. Чтобы получить объект класса для форматирования в национальном стандарте по умолчанию, используются следующие методы:

- NumberFormat.getlnstance()
- NumberFormat.getNumberlnstance()- идентичен getInstance()
- NumberFormat.getCurrencylnstance()
- NumberFormat.getPercentlnstance()

Чтобы получить объект класса для форматирования в других национальных стандартах используются следующие методы:

- NumberFormat.getlnstance(Locale locale)
- NumberFormat.getNumberlnstance(Locale locale) – идентичен методу getInstance(Locale locale)

Для значков валюты и процентов используются два метода класса NumberFormat:

- getCurrencylnstance(Locale locale)
- getPercentlnstance(Locale locale)

Метод getCurrencylnstance() задает добавление в вывод значка валюты, метод getPercentlnstance() значка процентов. Используется локаль для страны по умолчанию – locale. Вы в любой момент можете изменить локализацию для вывода значка валюты. Для работы с локализацией вам нужен класс

Листинг 3.7 – Пример использования NumberForma NumberForma и Locale

```
import java.text.NumberFormat;
import java.util.Locale;
public class NumberFormatDemo1 {
public static void main(String[] args) {
double number = 123.4567;
//определяем локализацию
Locale locFR = new Locale("fr");
Locale.setDefault(Locale.CHINA);
//определяем форматировщик 1
NumberFormat numberFormat1 =
NumberFormat.getInstance();
System.out.println("Число в текущей локали: " +
numberFormat1.format(number));
//определяем форматировщик 2
NumberFormat numberFormat2 =
NumberFormat.getInstance(Locale.US);
System.out.println("Число в китайской локали: " +
numberFormat2.format(number));
//определяем форматировщик 3
NumberFormat numberFormat3 =
NumberFormat.getCurrencyInstance();
System.out.println("Денежная единица в текущей
локали: " + numberFormat3.format(number));
//определяем форматировщик 4
NumberFormat numberFormat4 =
NumberFormat.getCurrencyInstance(Locale.FRANCE);
System.out.println("Денежная единица во французской
локали: " + numberFormat4.format(number));
//определяем форматировщик 5
NumberFormat numberFormat5 =
NumberFormat.getPercentInstance();
```

```
    System.out.println("Процент в текущей локали: " +
numberFormat5.format(number));
    //определяем форматировщик 6
NumberFormat numberFormat6 =
NumberFormat.getPercentInstance(locFR);
    System.out.println("Процент во французской локали: "
+ numberFormat6.format(number));
    }
}
```

Результат работы программы на листинге 3.8 представлен ниже:

```
Число в текущей локали: 123.457
Число в китайской локали: 123.457
Денежная единица в текущей локали: ¥123.46
Денежная единица во французской локали: 123,46 €
Процент в текущей локали: 12,346%
Процент во французской локали: 12 346 %
```

Задания на практическую работу № 3

Задания на Math и Random

1. Создать массив вещественных чисел случайным образом, вывести его на экран, отсортировать его, и снова вывести на экран (использовать два подхода к генерации случайных чисел – метод random() класса Math и класс Random)
2. Создать класс точка Point, описывающий точку на плоскости. Создать Circle класс, в котором одно поле представляет точку – центр окружности, и добавить другие свойства, позволяющие задать точку на плоскости. Создать третий класс Tester который использует для хранения объектов массив объектов Circle и второе поле количество элементов в массиве. Добавить в класс методы, позволяющие найти самую маленькую и самую большую окружность. Добавить в класс метод, упорядочивающий хранение окружностей в массив. Для инициализации полей радиуса и длины окружности используйте класс Random или метод random() класса Math по желанию
3. Создайте массив из 4 случайных целых чисел из отрезка [10;99], выведите его на экран в строку, далее определите и выведите на экран сообщение о том, является ли массив строго возрастающей последовательностью.
4. Пользователь должен ввести с клавиатуры размер массива - натуральное число больше, так чтобы введенное пользователем число сохранялось в переменную n. Если пользователь ввел не подходящее число, то

программа должна просить пользователя повторить ввод. Создать массив из n случайных целых чисел из отрезка [0; n] и вывести его на экран. Создать второй массив только из четных элементов первого массива, если они там есть, и вывести его на экран.

5. Пользователь должен ввести с клавиатуры размер массива - натуральное число больше, так чтобы введенное пользователем число сохранялось в переменную n. Если пользователь ввел не подходящее число, то программа должна просить пользователя повторить ввод. Создать массив из n случайных целых чисел из отрезка [0; n] и вывести его на экран. Создать второй массив только из четных элементов первого массива, если они там есть, и вывести его на экран

Задания на классы Оболочки

Задание 1

1. Создайте объекты класса Double, используя методы valueOf().
2. Преобразовать значение типа String к типу double. Используем метод Double.parseDouble().
3. Преобразовать объект класса Double ко всем примитивным типам.
4. Вывести значение объекта Double на консоль.
5. Преобразовать литерал типа double к строке: String d = Double.toString(3.14);

Задание 2

Заполнить таблицу табл. 3.2 Методы классов оболочек - на пересечении указать x, если данный метод существует у соответствующего класса оболочки.

Таблица 3.2 Методы классов оболочек

	Boolean	Byte	Character	Double	Float	Integer	Long	Short	isStatic
byteValue									
doubleValue									
floatValue									
intValue									
longValue									
shortValue									

Окончание табл. 3.2

	Boolean	Byte	Character	Double	Float	Integer	Long	Short	isStatic
parseXxx									

parseXxx with radix									
valueOf with radix									
toString									
toString(primitive)									
toString(primitive,radix)									

Задание на форматирование строк вывода

1. Создать класс конвертор валют.
2. Создать мини-Приложение интернет-магазин. В приложении рассчитывается стоимость покупки товара, где пользователь может выбрать валюту для оплаты товара
3. Разработать класс Отчет о сотрудниках

   1) Создать класс Employee, у которого есть переменные класса - fullname, salary.
   2) Создать массив, содержащий несколько объектов этого типа.
   3) Создать класс Report со статическим методом generateReport, в котором выводится информация о зарплате всех сотрудников.
   4) Используйте форматирование строк. Пусть salary будет выровнено по правому краю, десятичное значение имеет 2 знака после запятой и можете добавить что-нибудь свое.

Практическая работа № 4. Перечисления и их использование в Джава программах

Цель данной практической работы – познакомиться с новым ссылочным типом данных перечислением, научиться разрабатывать перечисления и использовать их в своих программах.

Теоретические сведения

Перечисления это один из объектных типов в Джава. Они являются безопасными типами, поскольку переменная тип перечисление может принимать значение только константу из перечисления. Рассмотрим пример объявления перечисления в языке Джва:

```
public enum Level {
    HIGH,
    MEDIUM,
    LOW
}
```

Обратите внимание на enum - ключевое слово перед именем перечисления, которое используется вместо class или interface. Ключевое слово enum сигнализирует компилятору Java, что это определение типа является перечислением. Вы можете ссылаться на константы в приведенном выше перечислении следующим образом:

```
Level level = Level.HIGH;
```

В этом случае переменная level принимает значение HIGH.

Обратите внимание, что level переменная имеет тип, Level который является типом перечисления Java, определенным в приведенном выше примере. Level.

Переменная может принимать одно значение из констант перечисления Level, то есть в качестве значения может принимать значения HIGH, MEDIUM, или LOW.

Использование перечислений в операторах ветвления if () else

Поскольку все перечисления в Джава являются константами, то вам часто придется сравнивать переменную, указывающую на константу перечисления, с возможными константами в типе перечисления. Вот пример использования перечисления в операторе if () else:

```
Level level = ...
//Присваиваем некоторое значение из перечисления Level
if( level == Level.HIGH) {
```

```
} else if( level == Level.MEDIUM) {
} else if( level == Level.LOW) {
}
```

В этом коде переменную level сравнивается с каждой из возможных констант перечисления в перечислении Level. Если оно соответствует одному из значений в перечислении, то проверка этого значения в первом операторе if приведет к повышению производительности, так как в среднем выполняется меньше сравнений.

Использование перечислений в операторах множественного выбора switch

Если ваши типы перечисления Java содержат много констант и вам нужно проверять переменную на соответствие нескольким значениям из перечисления, то воспользуйтесь оператором идеей switch.

Вы можете использовать перечисления в операторах switch следующим образом:

```
Level level = ...
// присвоить ему некоторую константу Level

switch (level) {
    case HIGH: ...;break;
    case MEDIUM: ...; break;
    case LOW: ...; break;
}
```

Замените многоточие ... в коде выше на то значение, которое вам нужно для выполнения кода, если переменная level истинно, то выполнится соответствующая ветка switch, т.е. в выражении переменная совпадет со значением константы соответствует заданному значению константы присвоенному level. Код, который соответствует определенной ветви switch может быть просто блок выражений или например может выполняться вызов метода и т.д.

Итерация перечислений

Вы можете получить массив всех возможных значений типа перечисления Джава, вызвав его статический метод values(). Все типы перечислений автоматически получают статический метод values() с помощью компилятора Джава. Вот пример итерации всех значений перечисления:

```
for (Level level : Level.values()) {
    System.out.println(level);
```

```
}
```

Запуск кода выше распечатает все значения перечисления.

Вот результат выполнения этого кода:

```
HIGH
MEDIUM
LOW
```

Обратите внимание, как печатаются имена самих констант. Это одно из отличий работы с перечислениями, перечисления Java отличаются от static final констант.

Использование метода toString () с перечислениями

Класс перечисления автоматически получает метод toString()при компиляции. Метод toString() возвращает строковое значение для данного экземпляра перечисления.

Пример:

```
Строка levelText = Level.HIGH.toString ();
```

Значением переменной levelText после выполнения вышеуказанного оператора будет текст HIGH.

Печать перечислений

Вы можете вывести на значение перечисления обычным способом, с помощью метода println(), например таким образом:

```
System.out.println (Level.HIGH);
```

Затем будет вызван метод toString(), поэтому значение, которое будет распечатано, будет именем экземпляра перечисления - текст. Другими словами, в приведенном выше примере будет напечатан текст HIGH.

Использование метода valueOf () с перечислениями

Класс перечисление автоматически получает статический метод valueOf() в классе при компиляции. Метод valueOf() может быть использован для получения экземпляра класса перечислимого для заданного значения String. Вот пример:

```
Level level = Level.valueOf ("BHIGH");
```

Переменная типа Level будет указывать на Level.HIGH после выполнения этой строки.

Поля перечислений

Вы можете добавлять поля данных в перечисление Джава, также как в классы. Таким образом, каждое константа перечисления получает эти поля.

Значения полей должны быть переданы конструктору перечисления при определении констант. Пример приведен на листинге 4.1.

Лстинг 4.1 – Пример перечисления Level

```
public enum Level {
    HIGH(3),  //вызов конструктора со значением 3
    MEDIUM(2),  // вызов конструктора со значением 2
    LOW(1)   // вызов конструктора со значением 1
/* нужна точка с запятой, если после констант следуют
поля и методы*.
    ;
 //определение полей перечисления
    private final int levelCode;
    private Level(int levelCode) {
        this.levelCode = levelCode;
    }
}
```

Обратите внимание, как у перечисления в приведенном выше примере есть конструктор, который принимает параметр типа int. Конструктор перечисления устанавливает поле int. Когда определены постоянные значения перечисления, int значение передается конструктору перечисления. Конструктор для перечисления должен быть объявлен с модификатором private. Вы не можете использовать конструкторы с модификаторами public или protected для перечислений в Джава. Если вы не укажете модификатор доступа в конструкторе перечисления, то он будет объявлен неявным образом с модификатором private.

Поля перечислений

Вы также можете добавлять методы в перечисление Джава. Пример приведен на листинге 4.2

Лстинг 4.2 – Пример перечисления Level с полями и методами

```
public enum Level {
  HIGH (3), // вызывает конструктор со значением 3
  MEDIUM (2),// вызывает конструктор со значением 2
  LOW (1) // вызывает конструктор со значением 1
    ; //здесь нужна точка с запятой
//поле для перечисления
  private final int levelCode;
//конструктор
  private Level (int levelCode) {
```

```
    this.levelCode = levelCode;
    }
// геттер для поля levelCode
     public int getLevelCode () {
         return this.levelCode;
     }
}
```

Вы вызываете метод перечисления Java через ссылку на одно из постоянных значений. Вот пример вызова метода перечисления Java:

```
Level level = Level.HIGH;
System.out.println (level.getLevelCode ());
```

Этот код распечатает значение, 3 которое является значением levelCode для поля HIGH константы перечисления

Кроме конструкторов, геттеров и сеттеров вы можете добавлять свои собственные методы в классы перечисления, которые будут производить вычисления на основе значений полей константы перечисления. Если ваши поля не объявлены как final, вы даже можете изменить значения полей (хотя это может быть не очень хорошая идея, учитывая, что перечисления должны быть константами).

Задания на практическую работу № 4

Задание 1. Времена года

Создать перечисление, содержащее названия времен года.

1) Создать переменную, содержащую ваше любимое время года и распечатать всю информацию о нем.
2) Создать метод, который принимает на вход переменную созданного вами enum типа. Если значение равно Лето, выводим на консоль “Я люблю лето” и так далее. Используем оператор switch.
3) Перечисление должно содержать переменную, содержащую среднюю температуру в каждом времени года.
4) Добавить конструктор, принимающий на вход среднюю температуру.
5) Создать метод getDescription, возвращающий строку “Холодное время года”. Переопределить метод getDescription - для константы Лето метод должен возвращать “Теплое время года”.
6) В цикле распечатать все времена года, среднюю температуру и описание времени года

Задание 2. Ателье

Создать перечисление, содержащее размеры одежды (XXS, XS, S, M, L). Перечисление содержит метод getDescription, возвращающий строку “Взрослый размер”. Переопределить метод getDescription - для константы XXS метод должен возвращать строку “Детский размер”. Также перечисление должно содержать числовое значение euroSize(32, 34, 36, 38, 40), соответствующее каждому размеру. Создать конструктор, принимающий на вход euroSize.

1) Создать интерфейсы MenClothing (мужская одежда) с методом dressMan() (одеть мужчину) и WomenClothing (женская одежда) с методом dressWomen() (одеть женщину).
2) Создать абстрактный класс Clothes (одежда), содержащий в качестве переменных класса - размер одежды, стоимость, цвет.
3) Создать классы наследники класса Clothes – класс TShirt (футболка) (реализует интерфейсы MenClothing и WomenClothing), класс Pants (штаны) (реализует интерфейсы MenClothing и WomenClothing), класс Skirt (реализует интерфейсы WomenClothing), класс Tie (галстук) (реализует интерфейсы MenClothing).
4) Создать массив, содержащий все типы одежды. Создать класс Atelier (Ателье), содержащий методы dressWomen, dressMan, на вход которых будет поступать массив, содержащий все типы одежды (подумайте какой тип будет у массива). Переопределите метод dressWomen() для вывода на консоль всей информации о женской одежде. То же самое сделайте для метода dressMan().

Задание 3. Интернет-магазин

Создать мини приложение - интернет-магазин. Должны быть реализованы следующие возможности:

1) Аутентификация пользователя. Пользователь вводит логин и пароль с клавиатуры.
2) Просмотр списка каталогов товаров.
3) Просмотр списка товаров определенного каталога.
4) Выбор товара в корзину.
5) Покупка товаров, находящихся в корзине.

Для выполнения заданий необходимо создать перечисление согласно заданию, можете добавить свои операции или изменить что-то по своему усмотрению.

Задание 4

Создать класс, описывающий сущность компьютер (Computer). Для описания составных частей компьютера использовать отдельные классы (Processor, Memory, Monitor). Описать необходимые свойства и методы для каждого класса. Для названий марок компьютера используйте перечисления (enum)

Практическая работа № 4.1. Наследование в Джава. Абстрактные классы.

Цель: познакомиться на практике с реализацией принципа ООП Наследование в Джава и освоить на практике работу с наследованием от абстрактных классов.

Теоретические сведения об абстрактных классах и наследовании

Наследование один из основных принципов разработки объектно-ориентированных программ. В терминологии Джава базовый класс, от которого производится наследование называется суперкласс. Производные классы, называются подклассы наследуют все компоненты родителей (поля и методы) кроме конструкторов и статических компонентов. Обратится к методам родительского класса можно, используя служебное слово super. Основное преимущество наследования — это возможность повторного использования кода. Наследование поддерживает концепцию «возможности повторного использования», т.е. когда мы хотим создать новый класс и уже существует класс, который включает в себя часть кода, который нам нужен, мы можем получить наш новый класс из уже существующего класса. Таким образом, мы повторно используем поля и методы уже существующего класса. Для организации наследования в Джава, используется ключевое слово – extends, что в переводе на русский означает расширяет. Таким образом создавая производный класс с помощью наследования, мы расширяем родительский класс как новыми свойствами – поля данных, так добавляем к нему новое поведение – методы класса. Пример на листинге 4.1.1

Листинг 4.1.1 – Пример наследования

```
// базовый класс велосипед
class Bicycle {
public int gear; // поле
public int speed; // поле

// конструктор класса
public Bicycle(int gear, int speed){
this.gear = gear;
this.speed = speed;
}
//метод класса
public void applyBrake(int decrement){
```

```
speed -= decrement;
}
//метод класса
public void speedUp(int increment){
speed += increment;
}
// метод toString()чтобы печатать объекты Bicycle
public String toString(){
return ("No of gears are " + gear + "\n"
+ "speed of bicycle is " + speed);
}
} // end of class

// производный класс горный велосипед
class MountainBike extends Bicycle {
public int seatHeight; //новое поле произв. класса
//конструктор производного класса
public MountainBike(int gear, int speed,
int startHeight){
// здесь вызов конструктора класса родителя
super(gear, speed);
seatHeight = startHeight;
}
// новый метод производного класса
public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }=
// переопределенный метод toString()класса Bicycle
@Override public String toString(){
return (super.toString() + "\nseat height is "
+ seatHeight);
  }
}
// класс тестер Main
public class Main {
public static void main(String args[]){
// создаем объект родительского класса
```

```
Bicycle bl = new Bicycle(5,200);
System.out.println(bl.toString());

// создаем объект дочернего класса
MountainBike mb = new MountainBike(3, 100, 25);
System.out.println(mb.toString());
    }
}
```

Абстрактные классы

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы при определении помечаются ключевым словом abstract.

Абстрактный метод внутри абстрактного класса не имеет тела, только прототип. Он состоит только из объявления и не имеет тела:

```
abstract void yourMethod();
```

По сути, мы создаём шаблон метода. Например, можно создать абстрактный метод для вычисления площади фигуры в абстрактном классе Фигура. А все другие производные классы от главного класса могут уже реализовать свой код для готового метода. Ведь площадь у прямоугольника и треугольника вычисляется по разным алгоритмам и универсального метода не существует.

Если вы объявляете класс, производный от абстрактного класса, но хотите иметь возможность создания объектов нового типа, вам придётся предоставить определения для всех абстрактных методов базового класса. Если этого не сделать, производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом **abstract**.

Абстрактный класс не может содержать какие-либо объекты, а также абстрактные конструкторы и абстрактные статические методы. Любой подкласс абстрактного класса должен либо реализовать все абстрактные методы суперкласса, либо сам быть объявлен абстрактным. Пример приведен на листинге 4.1.2

Листинг 4.1.2 – Пример абстрактного класса Swim

```
public abstract class Swim {
// абстрактный метод
abstract void swim()
// абстрактный класс может содержать иобычный метод
void run() {
System.out.println(“Куда идешь?”);
```

```
}
}
//создаем производный класс Swimmer
class Swimmer extends Swim {
….
}
```

Задания на практическую работу № 4.1

1. Необходимо реализовать простейший класс Shape (Фигура). Добавьте метод класса getType() (тип фигуры, возвращает строку тип String название фигуры). С помощью наследования создайте дочерние классы Circle, Rectangle и Square. (из предыдущей практической работы). Также реализуйте во всех классах методы getArea()(возвращает площадь фигуры), getPerimeter() взвращает периметр фигуры). Переопределите в дочерних класс методы класса родителя toString(), getArea(), getPerimeter() и getType(). Создать класс-тестер для вывода информации об объекте и продемонстирировать вызов методов использую родительскую ссылку. Объяснить работу программы.

2. Создайте класс Phone, который содержит переменные number, model и weight.

1)Создайте три экземпляра этого класса. 2) Выведите на консоль значения их переменных. 3) Добавить в класс Phone методы: receiveCall, имеет один параметр – имя звонящего. 4)Выводит на консоль сообщение “Звонит {name}”. 5)Метод getNumber – возвращает номер телефона. 6) Вызвать эти методы для каждого из объектов. 7) Добавить конструктор в класс Phone, который принимает на вход три параметра для инициализации переменных класса - number, model и weight. 8)Добавить конструктор, который принимает на вход два параметра для инициализации переменных класса - number, model. 9)Добавить конструктор без параметров. 10)Вызвать из конструктора с тремя параметрами конструктор с двумя. 11)Добавьте перегруженный метод. receiveCall, который принимает два параметра - имя звонящего и номер телефона звонящего. 12)Вызвать этот метод. 13)Создать метод sendMessage с аргументами переменной длины. Данный метод принимает на вход номера телефонов, которым будет отправлено сообщение. 14)Метод выводит на консоль номера этих телефонов.

3. Создать класс Person, который содержит: а) поля fullName, age. б) методы move() и talk(), в которых просто вывести на консоль сообщение -"Такой-то Person говорит". в) Добавьте два конструктора - Person() и Person(fullName, age). Создайте два объекта этого класса. Один объект инициализируется конструктором Person(), другой - Person(fullName, age).

4. Создать класс Матрица. Класс должен иметь следующие поля: 1) двумерный массив вещественных чисел; 2) количество строк и столбцов в матрице. Класс должен иметь следующие методы: 1) сложение с другой матрицей; 2) умножение на число; 3) вывод на печать; 4) умножение матриц - по желанию.

5. Класс «Читатели библиотеки». Определить класс Reader, хранящий такую информацию о пользователе библиотеки: ФИО, номер читательского билета, факультет, дата рождения, телефон. Методы takeBook(), returnBook(). Разработать программу, в которой создается массив объектов данного класса. Перегрузить методы takeBook(), returnBook(): - takeBook, который будет принимать количество взятых книг. Выводит на консоль сообщение "Петров В. В. взял 3 книги". - takeBook, который будет принимать переменное количество названий книг. Выводит на консоль сообщение "Петров В. В. взял книги: Приключения, Словарь, Энциклопедия". - takeBook, который будет принимать переменное количество объектов класса Book (создать новый класс, содержащий имя и автора книги). Выводит на консоль сообщение “Петров В. В. взял книги: Приключения, Словарь, Энциклопедия”. Аналогичным образом перегрузить метод returnBook(). Выводит на консоль сообщение “Петров В. В. вернул книги: Приключения, Словарь, Энциклопедия”. Или “Петров В. В. вернул 3 книги”.

6. Создайте пример наследования, реализуйте класс Employer и класс Manager. Manager отличается от Employer наличием дополнительных выплат от продаж а) Класс Employer содержит переменные: String firstName, lastName и поле income для заработной платы. Класс Manager также имеет поле double averageSum содержащую среднюю суммы дополнительных выплат за продажи. б) Создать переменную типа Employer, которая ссылается на объект типа Manager. в) Создать метод getIncome() для класса Employer, который возвращает заработную плату. Если средняя количество отработанных дней, то сумма дохода умножается на 12. Переопределить этот метод в классе Manager и добавить к доходу сумму с продаж. г) Создать массив типа Employer содержащий объекты класса Employer и Manager. Вызвать метод getIncome() для каждого элемента массива.

7. Создать суперкласс Учащийся и подклассы Школьник и Студент. Создать массив объектов суперкласса и заполнить этот массив объектами. Показать отдельно студентов и школьников.

Задания на абстрактные классы

8. Перепишите суперкласс Shape из задания 1, сделайте его

абстрактным и наследуйте подклассы, так как это представлено на UML диаграмме на рис. 4.1.1 Circle, Rectangle и Square.

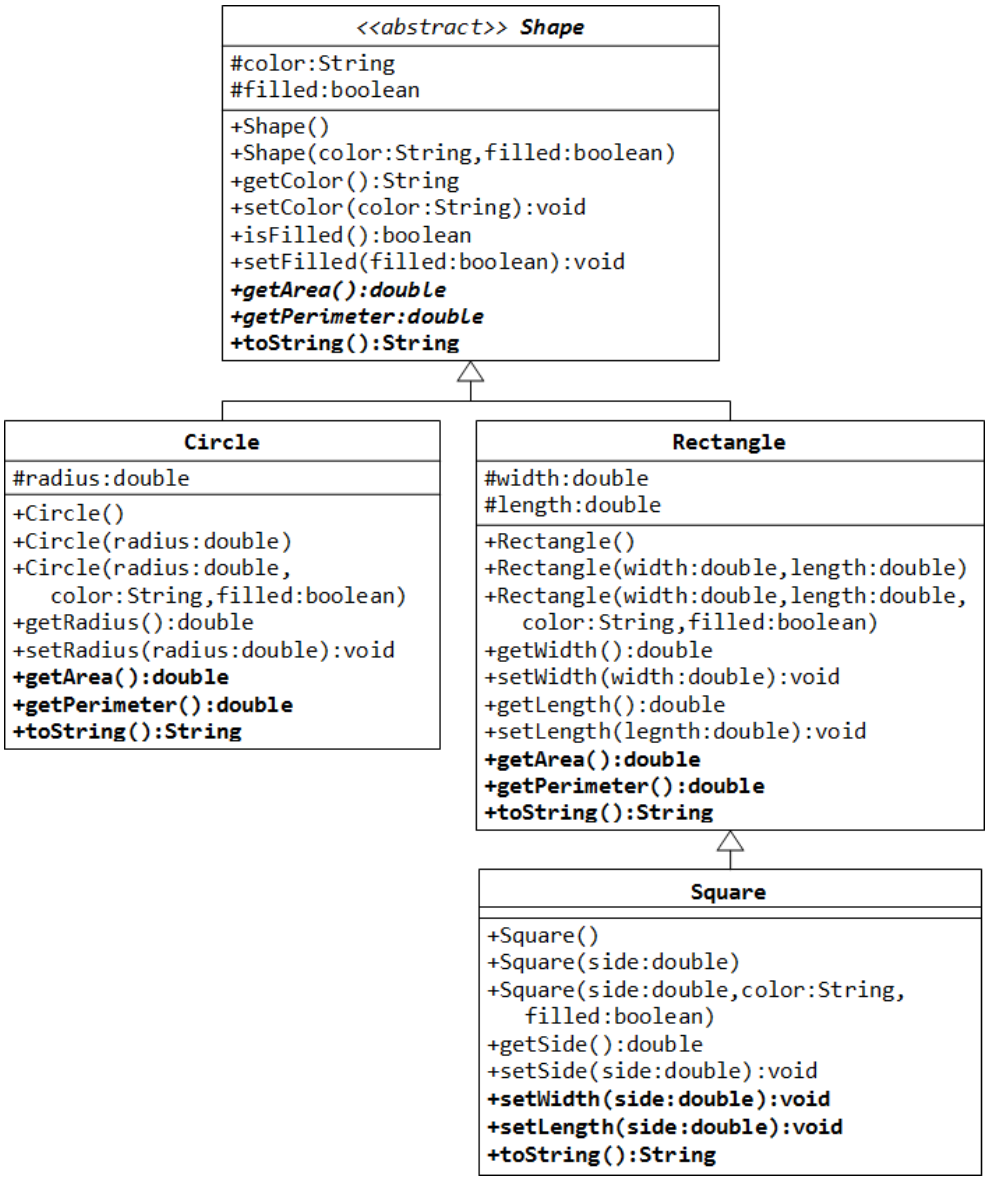

Рисунок 4.1.1. Диаграмма суперкласса Shape.

Замечания. В этом задании, класс Shape определяется как абстрактный класс, которыйсодержит:

- Два поля или переменные класса, объявлены с модификатором ***protected*** color (тип String) и filled (тип boolean). Такие защищенные переменные могут быть доступны в подклассах и классах в одном пакете. Они обозначаются со знаком “#” на диаграмме классов в нотации языка UML.
- Методы геттеры и сеттеры для всех переменных экземпляра класса,и метод toString().
- Два абстрактных метода getArea() и getPerimeter() выделены курсивом в диаграмме класса).

В подклассах Circle (круг) и Rectangle (прямоугольник) должны

переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс. Также необходимо для каждого подкласса переопределить toString() .

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

10. Создать абстрактный класс, описывающий Транспортное средство и подклассы Автомобиль, Самолет, Поезд, Корабль. Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.

Упражнение 1

Внимание. Внимание в заданиях есть код, который потенциально содержит ошибки, вам нужно объяснить ошибки и представить рабочую версию кода.

Вам нужно написать тестовый класс, чтобы самостоятельно это проверить, необходимо объяснить полученные результаты и связать их с понятием ООП - полиморфизм. Некоторые объявления могут вызвать ошибки компиляции. *Объясните полученные ошибки*, если таковые имеются.

Листинг 4.1.3 – Пример для выполнения

```
Shape s1 = new Circle(5.5, “RED”, false); // Upcast Circle to ShapeSystem.out.println(s1);//which version?
    System.out.println(s1.getArea());//  which  version?
System.out.println(s1.getPerimeter());//which    version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle  c1  =  (Circle)s1;//  Downcast  back  to  Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());
Shape s2 = new Shape();
Shape s3 = new Rectangle(1.0, 2.0, "RED", false);// Upcast
System.out.println(s3);
```

```
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3; //downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());
Shape s4 = new Square(6.6);//Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());
/*обратите внимание, что выполняем downcast Shape s4 к
Rectangle,      который      является      суперклассом
Square(родителем), вместо Square*//
Rectangle r2 = (Rectangle)s4;
System.out.println(r2); System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());
// Downcast Rectangle r2 к Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Практическая работа № 5. Создание программ с графическим интерфейсом пользователя на языке Джава

Цель: цель данной практической работы – научится разрабатывать программы на языке Джава с использованием графического интерфейса пользователя.

Теоретические сведения

Swing в Джава — это набор инструментов графического интерфейса пользователя (GUI), который включает компоненты GUI. Swing предоставляет богатый набор виджетов и пакетов для создания сложных компонентов графического интерфейса пользователя для приложений Java. Swing является частью Java Foundation Classes (JFC), который представляет собой API для программирования Java GUI, который предоставляет GUI. Библиотека Java Swing построена на основе Java Abstract Widget Toolkit (AWT), более старого, зависящего от платформы инструментария графического интерфейса пользователя. Вы можете использовать простые программные компоненты Java с графическим интерфейсом пользователя, такие как кнопка, текстовое поле и т. д. Из библиотеки, и вам не нужно создавать компоненты с нуля. Схема иерархии классов графической библиотеки Swing представлена на рис. 5.1.

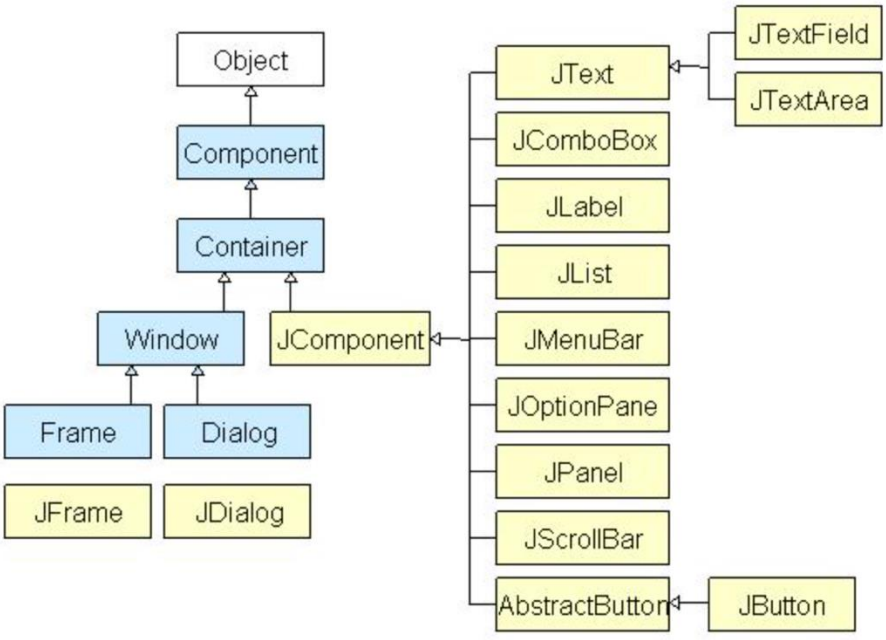

Рисунок 5.1. Схема иерархии классов Swing

Все компоненты в Java Swing — это JComponent, который можно добавлять в классы-контейнеры.

Классы-контейнеры

Классы-контейнеры — это классы, на которых могут быть другие компоненты. Итак, для создания графического интерфейса Java Swing нам понадобится хотя бы один объект-контейнер. Существует три типа контейнеров Java Swing:

- JPanel (Панель): это чистый контейнер, а не окно. Единственная цель Panel - организовать компоненты в окне.
- JFrame (Фрейм): это полностью функционирующее окно со своим заголовком и значками.
- JDialog (Диалог): это можно представить как всплывающее окно, которое выскакивает, когда необходимо отобразить сообщение. Это не полностью функционирующее окно, как Frame.

Рассмотрим последовательно все шаги по созданию графического интерфейса на Джава с помощью Swings

Листинг 5.1 – Пример 1 программы с GUI

```
import javax.swing.*;
class FirstGui{
public static void main(String args[]){
//создаем фрейм окна с помощью конструктора
//Конструктор берет параметр – назаъвание окна – это строка
JFrame frame = new JFrame("My First GUI");
// устанавливаем реакцию окна на закрытие по умолчанию
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//задаем свойства окна – его размеры в пискселях
 frame.setSize(300,300);
//создаем кнопку с помощью конструктора класса JButton
//конструктор берет параметр строку – название на кнопке
JButton button = new JButton("Press");
//добавляем кнопку ук окну
frame.getContentPane().add(button);
//делаем окно видимым
frame.setVisible(true);
    }
}
```

На рис. 5.2 показан результат работы программы на листинге 5.1

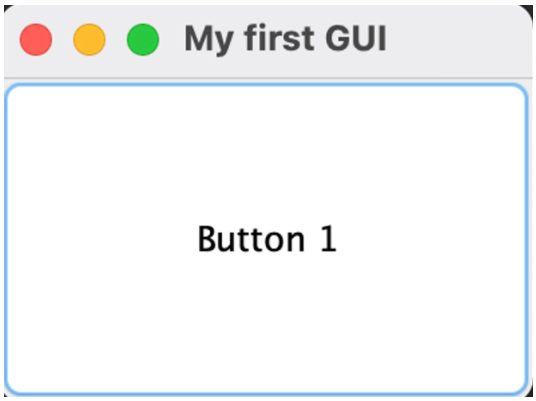

Рисунок 5.2. Окно графического интерфейса программы 5.2

Листинг 5.2 – Пример 2 программы с GUI

```
import javax.swing.*;
class SecondGui{
public static void main(String args[]){
JFrame frame = new JFrame("My First GUI");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300,300);
JButton button1 = new JButton("Button 1");
//сощдали еще одну кнопку
JButton button2 = new JButton("Button 2");
frame.getContentPane().add(button1);
//добавили вторую кнопку
frame.getContentPane().add(button2);
frame.setVisible(true);
     }
}
```

На рис. 5.3 показан результат работы программы на листинге 5.2

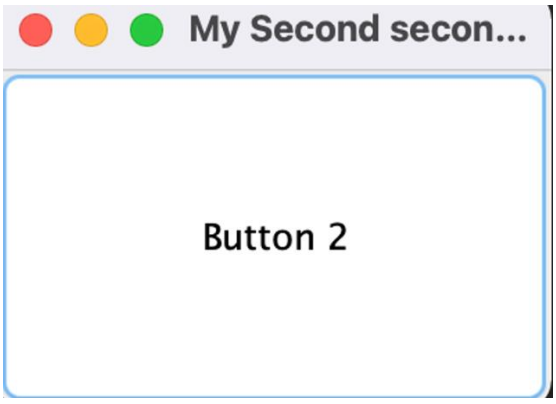

Рисунок 5.3. Окно графического интерфейса программы 5.3

Мы видим на рисунке только вторую кнопку. Дело в том, что приложение имеет многослойную структуру и следующий компонент, который мы добавили

закрывает или наслаивается н предыдущий. Для того чтобы избежать этого эффекта нужно использовать контейнеры для организации компонентов и менеджеры компоновки. Добавим к нашему коду объект класса JPanel, он использует менеджер компоновки по умолчанию. Но вы можете изменить его с помощью метода setLayout().

Листинг 5.3 – Пример 3 программы с GUI

```
import java.awt.*;
import javax.swing.*;
public class ThirdGui {
public static void main(String args[]) {
JFrame frame = new JFrame("My Second second GUI");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(200, 150);
//создали панель
JPanel panel= new JPanel();
//задали свойство панели – цвет фона
panel.setBackground(Color.GRAY);
// задали свойство панели размеры
panel.setPreferredSize( new Dimension(200,300));
JButton button1 = new JButton("Button 1");
JButton button2 = new JButton("Button 2");
panel.add(button1);
panel.add(button2);
//добавили панель к фрейму
frame.getContentPane().add(panel);
//упакуем во фрейм
frame.pack();
frame.setVisible(true);

    }
}
```

На рис. 5.4 показан результат работы программы на листинге 5.3. Два объекта класса JButton будут добавляться последовательно один за другим, том порядке в котором они были добавлены с помощью метода add в программе, поскольку в данном случае используется менеджер компоновки по умолчанию. Если вы хотите изменить расположение объектов используйте менеджеры компоновки и их сочетания. Изменить можно с помощью метода setLayout.

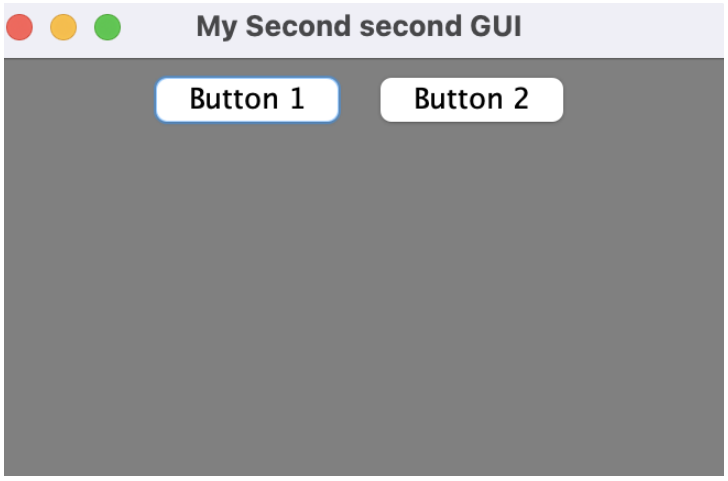

Рисунок 5.4. Окно графического интерфейса программы 5.3

Класс JLabel

JLabel это область для отображения короткой строки, изображения или того и другого. Обычно объекты JLabel добавляются на панели. При добавлении следующего кода Java после создания JFrame создается надпись с текстом “I'm a JLabel”. Пример кода:

```
JLabel label = new JLabel("I'm a JLabel", JLabel.CENTER);
frame.add(label);
```

Для позиции метки, он может быть указан JLabel.LEFT, JLabel.CENTER, JLabel.RIGHT, из которых позиция может быть установлена слева, по центру и справа correspondently. На рисунке ниже показано, что метка установлена в центре окна.

Мы получим результат как на рис. 5.5

Рисунок 5.5. Окно Программы с Jlabel

Задания на практическую работу № 5

1. Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Порядок работы: 1) Создайте пользовательское JFrame приложение, у которого есть следующие компоненты GUI:

- одна кнопка JButton подписана “AC Milan”
- другая JButton подписана “Real Madrid”
- надпись JLabel содержит текст “Result: 0 X 0”
- надпись JLabel содержит текст “Last Scorer: N/A”
- надпись Label содержит текст “Winner: DRAW”;

Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

2. 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.

3. Создать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки.

4. Создать окно, реализовать анимацию, с помощью картинки, состоящей из нескольких кадров.

Практическая работа № 6. Интерфейсы в Java

Цель: цель данной практической работы – научится разрабатывать практике пользовательские интерфейсы, и применять их в программах на языке Джава.

Теоретические сведения

Интерфейс в Java это разновидность класса. В качестве компонентов интерфейс имеет поля данных только статические константы и в качестве методов только абстрактные методы.

Интерфейс в Java — это механизм для достижения определенного рода абстракции. Интерфейсе Java не может включать никаких других методов кроме абстрактных. В составе интерфейса методы только объявлены, у них нет реализации или тела метода. Это похоже на механизм виртуальных функций в языке C++. Интерфейс содержит только объявления методов, то есть тело метода отсутствует. В Джава может быть объявлен пустой интерфейс, котрый не содержит ни одного объявления метода. Такие интерфейсы называются этикетками. Все методы входящие в интерфейс объявляются как abstract public, но вы можете не писать это перед самим методом, так как все методы, входящие в интерфейс и так по умолчанию абстрактные и с открытым - public доступом (начиная с версии Java 8). В Java 9 методы могут быть объявлены с модификатором private. Интерфейсы используются для достижения абстракции и множественного наследования в языке Джава. Потому что один и тот же интерфейс может использоваться для реализации разными классами.

Интерфейс Java также представляет отношение классами “IS- a”.

Преимущества интерфейсов

Существуют по крайней мере три веские причины использовать интерфейсы:

- они используется для достижения абстракции.
- Благодаря интерфейсам мы можем поддерживать механизм множественного наследования.
- они использовать для достижения слабой связанности кода (low coupling code)[4]

[4] Термин low coupling code означает слабую связанность, то есть необходимо распределить ответственности между классами так, чтобы обеспечить минимальную связанность в коде. https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/

Объявление интерфейсов

Интерфейс объявляется с помощью ключевого слова interface. Он обеспечивает полную абстракцию; это означает, что все методы в интерфейсе объявлены с пустым телом, а все поля по умолчанию являются общедоступными, статическими и окончательными (объявлены с модификатором final). Класс, реализующий интерфейс, должен реализовывать все методы, объявленные в интерфейсе.

Внимание! Если вы забудете реализовать какой-нибудь из методов, то вы получите абстрактный класс.

Синтаксис объявления интерфейса:

```
interface <interface_name> {

    // объявляем поля константы
    // объявляем абстрактные методы
    // (они по умолчанию абстрактные)
}
```

Замечание: Компилятор Java добавляет ключевые слова public и abstract перед методом интерфейса. Более того, он добавляет ключевые слова public, static и final перед полями данных класса.

Улучшение интерфейсов, начиная с Java 8

Начиная с Java 8, интерфейс может иметь методы по умолчанию и статические методы, которые обсуждаются позже.

Отношения между классами и интерфейсами при наследовании

Как показано на приведенном ниже рис. 6.1, класс может расширять другой класс при наследовании, интерфейс может расширять другой интерфейс при наследовании, но только класс реализует интерфейс. То есть в нем должны быть реализованы все методы интерфейса, который он при объявлении реализует.

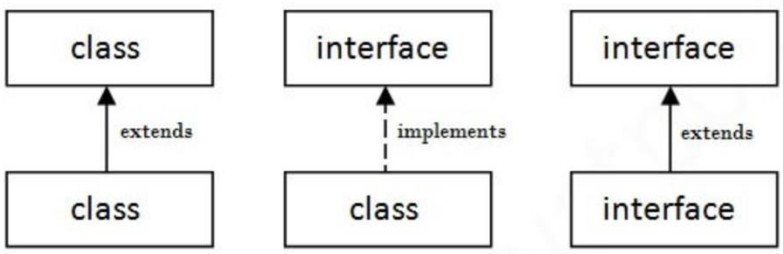

Рисунок 6.1. Схема отношений между классами

Пример интерфейса Java: реализация возможности рисования

В этом примере интерфейс Drawable имеет только один метод. Его реализация обеспечивается в классах Rectangle и Circle, котрые реализуют этот интерфейс. В реальном сценарии в жизни интерфейс определяется кем-то другим, но его реализация обеспечивается разными поставщиками реализации. Более того, им всегда пользуется кто-то другой. Часть реализации скрыта пользователем интерфейса, то есть классом, который и реализует интерфейс.

Листинг 6.1 – Пример работы с интерфейсом

```
// Объявление интерфейса: первым пользователем
interface  Drawable {
void  draw ();
}
// Реализация: вторым пользователем
class  Rectangle implements Drawable {
public void draw () {
System.out.println("Рисование прямоугольника\n" );
}}
class  Circle implements Drawable {
 public void draw(){System.out.println("Рисование круга\n"
);
}}
// Использование интерфейса: третьим пользователем
class  Main {
public static void main (String args []) {
Drawable d = new Circle ();
/* В реальном сценарии объект предоставляется методом,
например, getDrawable () */
d.draw ();
}}
```

На рис. 6.2 мы видим UML диаграмму трех классов и одного интерфейса. Как видно из отношений на схеме, все три класса реализуют общий интерфейс.

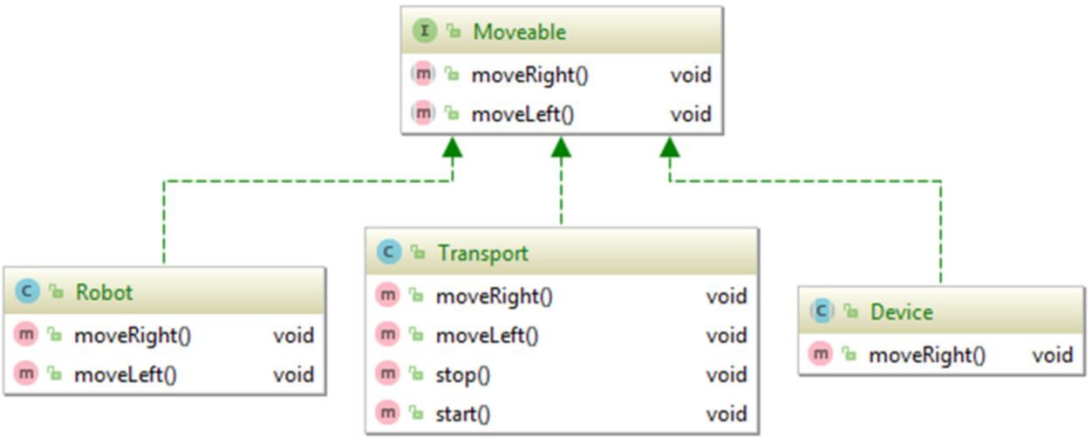

Рисунок 6.2. UML диаграмма классов

Из примера, приведенного в листинге 6.1 мы видим использование интерфейсной ссылки для инициализации объектном созданного класса. Один интерфейс в Джава может быть реализован множеством классов см. рисунок

Задания на практическую работу №6

1. Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable (см рис. 6.3)

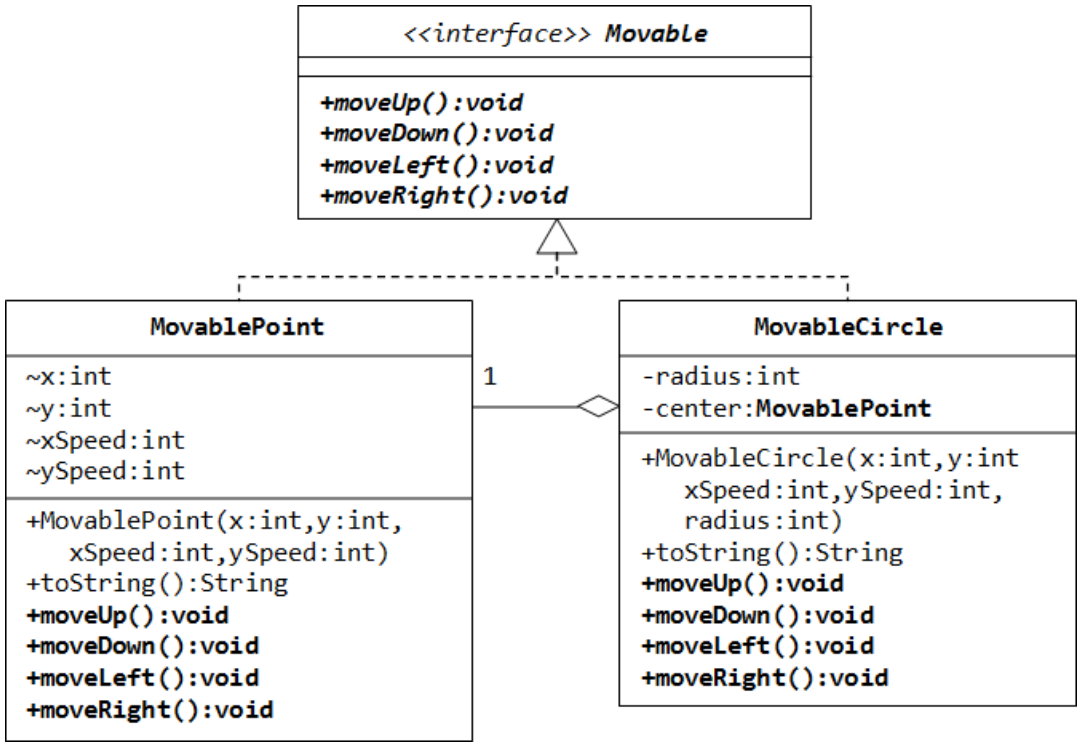

Рисунок 6.3. Диаграмма реализации итерфейса Movable.

```
public interface Movable {
// сохраните как "Movable.java"
public void moveUp(); //метод интерфейса
......
}
```

2. Напишите новый класс MovableRectangle (движущийся прямоугольник). Его можно представить как две движущиеся точки MovablePoints (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс Movable, см рис. 6.4. Убедитесь, что две точки имеет одну и ту же скорость (вам понадобится метод проверяющий это условие).

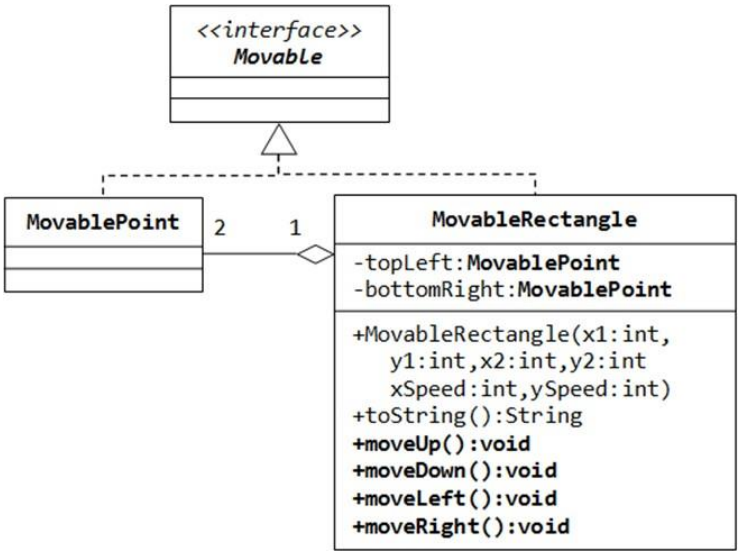

Рисунок 6.4. Диаграмма класса MovableRectangle.

3. 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.).

4. Реализовать интерфейс Priceable, имеющий метод getPrice(), возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

5. Вам нужно написать два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable на основе классов, разработанных в практической работе № 5. Изучите UML диаграмму и представьте реализацию класса Замечание. Создайте в draw.io UML диаграмму, а затем напишите по ней реализацию.

6. Определить интерфейс Printable, содержащий метод void print().

7. Определить класс Book, реализующий интерфейс Printable.

8. Определить класс Shop, реализующий интерфейс Printable.

9. Создать массив типа Printable, который будет содержать книги и журналы. В цикле пройти по массиву объектов и вызвать метод print() для каждого объекта.

10. Мини приложение Интернет-магазин компьютерной техники. Создать класс, описывающий сущность компьютер (Computer). Для описания

составных частей компьютера использовать отдельные классы (Processor, Memory, Monitor). Описать необходимые свойства и методы для каждого класса. Для названий марок компьютера используйте перечисления (enum). Разработайте класс Shop для, реализуйте методы добавления и удаления компьютеров в магазине, добавьте метод поиска в магазине компьютера, нужного пользователю. Протестируйте работу созданных классов. Данные для заполнения массива компьютеров вводятся с клавиатуры пользователем. Для этого реализуйте интерфейс.

11. Напишите программу для перевода температуры по Цельсию в температуру по Кельвину и Фаренгейту. Для этого добавьте интерфейс Convertable у которого есть метод convert для конвертации из одной системы измерения в другую.

12. Напишите свой класс StringBuilder с поддержкой операции undo. Для этого делегируйте все методы стандартному StringBuilder[5], а в собственном классе храните список всех операций для выполнения undo(). Данная программа пример реализации поведенческого шаблона «Команда»[6]. Пример можно посмотреть здесь https://refactoring.guru/ru/design-patterns/command/java/example.

13. Напишите свой класс StringBuilder, с возможностью оповещения других объектов об изменении своего состояния. Для этого делегируйте все методы стандартному StringBuilder, а в собственном классе реализуйте шаблон проектирования «Наблюдатель»[7]. Пример реализации можно посмотреть здесь https://refactoring.guru/ru/design-patterns/observer/java/example

14. Составьте отчет и представьте на проверку преподавателю

[5] https://docs.oracle.com/javase/10/docs/api/java/lang/StringBuilder.html

[6]https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

[7] https://ru.wikipedia.org/wiki/Наблюдатель_(шаблон_проектирования)#Java

Практическая работа №7. Реализация интерфейсов

Цель: цель данной практической работы – научится разрабатывать практике пользовательские интерфейсы, и применять их в программах на языке Джава.

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 Джава подобную проблему позволяют решить интерфейсы. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применитьмножество 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. Вданном 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. Вданном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, нофактически 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:

Листинг 7.1 – Пример 1 реализации интерфейса

```
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 иего метод print:

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

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

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

Одним из преимуществ использования 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:

Листинг 7.2 – Пример 2 реализации интерфейса

```
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();//для одного объекта
    printable = new Journal("Хакер");
    printable.print();//для другого объекта
```

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

Листинг 7.3 – Пример 3 реализации интерфейса

```
    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"));
    }
    //статический метод класса
    static void read(Printable p){
    p.print();
    }
    //статический метод
    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 можем возвратить как объект Book, так и Journal.

Статические методы интерфейса

Интерфейс Java может иметь статические методы. Статические методы в интерфейсе Java должны иметь реализацию, в отличие от обычных методов. Вот пример статического метода в интерфейсе Java:

Листинг 7.4 – Пример интерфейса со статическим методом

```
public interface MyInterface {

    public static void print (String text) {
        System.out.print (текст);
    }
}
```

Вызов статического метода в интерфейсе выглядит и работает так же, как вызов статического метода в классе. Вот пример вызова статического print() метода из MyInterface интерфейса выше :

```
MyInterface.print ("Привет, статический метод!");
```

Статические методы в интерфейсах могут быть полезны, когда у вас есть некоторые служебные методы, которые вы хотели бы сделать доступными, которые естественным образом вписываются в интерфейс, связанный с той же ответственностью.

Например, Vehicle интерфейс может иметь статический метод `printVehicle(Vehicle v)`.

Задания на практическую работу № 7

1. Создайте в draw.io UML диаграмму и напишите по ней реализацию. Диаграмма должна включать в себя следующие элементы: интерфейс Movable, содержащий в себе методы для движения прямоугольника (вверх, вниз, влево, вправо) и класс MovableRectangle (движущийся прямоугольник), реализующий интерфейс Movable.

2. Напишите по диаграмме класс MovableRectangle (движущийся прямоугольник), реализующий интерфейс Movable, класс прямоугольник, который можно представить как две движущиеся точки MovablePoint (верхняя левая и нижняя правая точки – topLeft и bottomRight), также реализующие интерфейс Movable;

3. Добавьте в класс параметризированные конструкторы, входящие в состав классов; метод в классах для перевода числовых значений в Строку. Убедитесь, что две точки имеют одну и ту же скорость при помощи специального логического метода SpeedTest(), проверяющего это.

4. Разработайте интерфейс MathCalculable, который содержит

объявления математических функций: возведения в степень и модуль комплексного числа, также содержит число PI. Напишите класс MathFunc, который реализует, реализует этот интерфейс. Например, вычисления длины окружности, для чего используйте число PI из интерфейса. Протестируйте класс

Замечание:

```
MathCalculable mc1 = new MathFunc(); // правильно
MathCalculable mc2 = new MathCalculable ();
//ошибка - запрещено объявлять экземпляр интерфейса
```

5. Разработайте интерфейс для работы со строками, который содержит а) функции подсчета символов в строке б) функция возвращает строку, которая образовывает строку, состоящую из символов исходной строки s, которые размещены на нечетных позициях: 1, 3, 5, ...в) функцию инвертирования строки

6. Реализуйте интерфейс в классе ProcessStrings и протестируйте работу класса

7. Создать статический метод printMagazines(Printable[] printable) в классе Magazine, который выводит на консоль названия только журналов.

8. Создать статический метод printBooks(Printable[] printable) в классе Book, который выводит на консоль названия только книг. Используем оператор instanceof.

9. Представьте отчет преподавателю на проверку

Практическая работа №8. Рекурсия Программирование рекурсии в Java. Решение задач на рекурсию

Цель: разработка и программирование рекурсивных алгоритмов на языке Java.

Теоретические сведения

В контексте языка программирования рекурсия — это некий активный метод (или подпрограмма) вызываемый сам по себе непосредственно, или вызываемой другим методом (или подпрограммой) косвенно. В первую очередь надо понимать, что рекурсия — это своего рода перебор. Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции.

Так же, как и у перебора (цикла) у рекурсии должно быть условие остановки — базовый случай (иначе также, как и цикл, рекурсия будет работать вечно — infinite). Это условие и является тем случаем, к которому рекурсия идет (шаг рекурсии). При каждом шаге вызывается рекурсивная функция до тех пор, пока при следующем вызове не сработает базовое условие и не произойдет остановка рекурсии (а точнее возврат к последнему вызову функции). Всё решение сводится к поиску решения для базового случая. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор, пока не получим базовое решение.

Итак, рекурсивная функция состоит из:

- условие остановки или же *базового случая* или условия;
- условие продолжения или *шага рекурсии* — способ сведения сложной задачи к более простым подзадачам.

Рассмотрим это на примере нахождения факториала:

Листинг 8.1 – Пример вычисления факториала

```
public class Solution {
public static int recursion(int n) {
// условие выхода
// Базовый случай
// когда остановиться повторять рекурсию ?if (n == 1)
{
return 1;
}
// Шаг рекурсии / рекурсивное условие
return recursion(n - 1) * n;
```

```
    }
    public static void main(String[] args) {
System.out.println(recursion(5)); // вызов рекурсивной функции
    }}
```

Тут базовым условием является условие, когда n=1. Так как мы знаем, что 1!=1 и для вычисления 1! нам ни чего не нужно. Чтобы вычислить 2! мы можем использовать 1!, т.е. 2!=1!*2. Чтобы вычислить 3! нам нужно 2!*3… Чтобывычислить n! нам нужно (n-1)!*n. Это и является шагом рекурсии.

Иными словами, чтобы получить значение факториала от числа n, достаточно умножить на n значение факториала от предыдущего числа.

В сети интернет при объяснении понятия рекурсии часто даются примеры решения задач нахождения чисел Фибоначчи и Ханойская башня

Представлены задачи с различным уровнем сложности. Попробуйте их решить, самостоятельно используя подход, описанный выше.

При решении попробуйте думать рекурсивно и ответить на вопросы:

- какой базовый случай или условие задается в задаче?
- какой шаг рекурсии или рекурсивное условие?

Задания на практическую работу № 8

Замечание. Задания по этой теме выполняются следующим образом: каждый учащийся выполняет от 3 до 5 задач, начиная с номера варианта задания, который соответствует номеру учащегося в журнале группы.

1. Задание Треугольная последовательность

Дана монотонная последовательность, в которой каждое натуральное число k встречается ровно k раз: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4,…

По данному натуральному n выведите первые n членов этой последовательности. Попробуйте обойтись только одним циклом for.

2. Задание от 1 до n

Дано натуральное число n. Выведите все числа от 1 до n.

3. Задание от A до B

Даны два целых числа A и B (каждое в отдельной строке). Выведите все числа от A до B включительно, в порядке возрастания, если A < B, или в порядке убывания в противном случае.

4. Задание Заданная сумма цифр

Даны натуральные числа k и s. Определите, сколько существует k-значных натуральных чисел, сумма цифр которых равна d. Запись натурального числа не может начинаться с цифры 0.

В этой задаче можно использовать цикл для перебора всех цифр, стоящих

на какой-либо позиции.

5. Задание Сумма цифр числа

Дано натуральное число N. Вычислите сумму его цифр. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).

6. Задание Проверка числа на простоту

Дано натуральное число n>1. Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое и NO, если число составное. Алгоритм должен иметь сложность O(logn).

Указание. Понятно, что задача сама по себе нерекурсивна, т.к. проверка числа n на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно сделать еще один параметр рекурсии: делитель числа, и именно по этому параметру и делать рекурсию.

7. Задание Разложение на множители

Дано натуральное число n>1. Выведите все простые множители этого числа в порядке не убывания с учетом кратности. Алгоритм должен иметь сложность O(logn)

8. Задание Палиндром

Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO.

При решении этой задачи нельзя пользоваться циклами, в решениях на питоне нельзя использовать срезы с шагом, отличным от 1.

9. Задание Без двух нулей

Даны числа a и b. Определите, сколько существует последовательностей из a нулей и b единиц, в которых никакие два нуля не стоят рядом.

10. Задание Разворот числа

Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке.

При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика.

Функция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

11. Задание Количество единиц

Дана последовательность натуральных чисел (одно число в строке), завершающаяся двумя числами 0 подряд. Определите, сколько раз в этой последовательности встречается число 1. Числа, идущие после двух нулей, необходимо игнорировать.

В этой задаче нельзя использовать глобальные переменные и параметры,

передаваемые в функцию. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметров.

12. Задание Вывести нечетные числа последовательности

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите все нечетные числа из этой последовательности, сохраняя их порядок.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

13. Задание Вывести члены последовательности снечетными номерами

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите первое, третье, пятое и т.д. из введенных чисел. Завершающий ноль выводить не надо.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

14. Задание Цифры числа слева направо

Дано натуральное число N. Выведите все его цифры по одной, в обычном порядке, разделяя их пробелами или новыми строками.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика

15. Задание Цифры числа справа налево

Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика.

16. Задание Количество элементов, равных максимуму

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом ноль. Определите, какое количество элементов этой последовательности, равны ее наибольшему элементу.

В этой задаче нельзя использовать глобальные переменные. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметра. В программе на языке Python функция возвращает результат в виде кортежа из

нескольких чисел, и функция вообще не получает никаких параметров. В программе на языке C++ результат записывается в переменные, которые передаются в функцию по ссылке. Других параметров, кроме как используемых для возврата значения, функция не получает. Гарантируется, что последовательность содержит хотя бы одно число(кроме нуля)

17. Задание Максимум последовательности

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Определите наибольшее значение числа в этой последовательности.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция возвращает единственное значение: максимум считанной последовательности. Гарантируется, что последовательность содержит хотя бы одно число (кроме нуля).

Пример решения задачи

Задание: найти точную степень двойки. Дано натуральное число N. Выведите слово YES, если число N является точной степенью двойки, или слово NO в противном случае.

Решение:

```
public class Rec1 {
public static int recursion(double n) { if (n == 1) {
return 1;
}
else if (n > 1 && n < 2) { return 0;
}
else {
return recursion(n / 2);
}
}
public static void main(String[] args) { double n = 64;

if (recursion(n) == 1) { System.out.println("Yes");
} else {
System.out.println("No");
}
}
}
```

Практическая работа № 9. Использование полиморфизма при программировании при реализации алгоритмов сортировок и поиска

Цель работы: освоение на практике методов сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

Теоретические сведения

Сортировка — это процесс упорядочивания списка элементов (организация в определенном порядке) исходного списка элементов, который возможно организован в виде контейнера или храниться в виде коллекции.

Процесс сортировки основан на упорядочивании конкретных значений, например:

- сортировка списка результатов экзаменов баллов в порядке возрастания результата;
- сортировка списка людей в алфавитном порядке по фамилии.

Есть много алгоритмов для сортировки списка элементов, которые различаются по эффективности.

Алгоритм сортировки вставками.

Работа метода сортировки состоит из следующих шагов:

- выбрать любой элемент из списка элементов и вставить его в надлежащееместо в отсортированный подсписок;
- повторять предыдущий шаг, до тех пор, пока все элементы не будут вставлены.

Более детально:

- рассматриваем первый элемент списка как отсортированный подсписок (то есть первый элемент списка);
- вставим второй элемент в отсортированный подсписок, сдвигая первый элемент по мере необходимости, чтобы освободить место для вставки нового элемента;
- вставим третий элемент в отсортированный подсписок (из двух элементов), сдвигая элементы по мере необходимости;
- повторяем до тех пор, пока все значения не будут вставлены на свои соответствующие позиции.

Алгоритм быстрой сортировки (Quick Sort).

Состоит из последовательного выполнения двух шагов:

- массив A[1..n] разбивается на два непустых подмассивов по

отношению к"опорному элементу”;

- два подмассива сортируются рекурсивно посредством Quick Sort.

Алгоритм сортировка слиянием (Merge Sort).

Состоит из последовательного выполнения трех шагов:

- разделить массив A[1..n] на 2 равные части;
- провести сортировку слиянием двух подмассивов (рекурсивно);
- объединить (соединить) два отсортированных подмассива.

Использование интерфейса Comparable в Джава программах для сортировки объектов

Техника программирования сортировок в Джава отличается от написания алгоритмов на процедурных языках программирования. При написании кода большим преимуществом является использование основного принципа ООП – полиморфизма. Напомним, что класс, который реализует интерфейс Comparable определяет метод compareTo(), чтобы определить относительный порядок своих объектов.

Таким образом мы можем использовать полиморфизм, чтобы разработать обобщенную сортировку для любого набора Comparable объектов.

При разработке класса, реализующего метод сортировки, нужно помнить, что метод принимает в качестве параметра массив объектов типа Comparable или фактически полиморфных ссылок.

Таким образом, один метод, может быть, использован для сортировки любых объектов, например: People (людей), Books (книг), или любой каких-либо других объектов.

Методу сортировки все-равно, что именно он будет сортировать, ему только необходимо иметь возможность вызвать метод compareTo().

Это обеспечивается использованием в качестве типа формального параметра интерфейса Comparable или интерфейсной ссылки.

Кроме того, таким образом каждый класс “для себя” решает, что означает для одного объекта, быть меньше, чем другой.

Листинг 9.1 – Пример решения задачи.

```
public class Sorting {
public static void selectionSort (Comparable[] list){
int min;
Comparable temp;
for (int index = 0; index < list.length-1; index++){
min = index;
for (int scan = index+1; scan < list.length; scan++)
```

```
 if (list[scan].compareTo(list[min]) < 0){
  min = scan;
  temp = list[min];
  list[min] = list[index];
  list[index] = temp;
    }
  }
}
```

Задания на практическую работу № 9

1. Написать тестовый класс, который создает массив класса Student и сортирует массив iDNumber и сортирует его вставками.

2. Напишите класс SortingStudentsByGPA который реализует интерфейс Comparator таким образом, чтобы сортировать список студентов по их итоговым баллам в порядке убывания с использованием алгоритма быстрой сортировки.

3. Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках с использованием алгоритма сортировки слиянием.

4. Напишите свою собственную реализацию интерфейса Comparable

Практическая работа № 10. Стандартные интерфейсы Джава. Интерфейс Comparator

Цель: цель данной практической работы - закрепить знания в области использования стандартных интерфейсов языка Джава, научиться применять интерфейсы для разработки практических программ на Джаве

Теоретические сведения

Comparator и Comparable в Java

Интерфейс Comparable содержит один единственный метод int compareTo(E item), который сравнивает текущий объект с объектом, переданным в качестве параметра. Если этот метод возвращает отрицательное число, то текущий объект будет располагаться перед тем, который передается через параметр. Если метод вернет положительное число, то, наоборот, после второго объекта. Если метод возвратит ноль, значит, оба объекта равны.

Интерфейс Comparator

Однако перед нами может возникнуть проблема, что, если разработчик не реализовал в своем классе, который мы хотим использовать, интерфейс Comparable, либо реализовал, но нас не устраивает его функциональность, и мы хотим ее переопределить? На этот случай есть еще более гибкий способ, предполагающий применение интерфейса Comparator<E>.

Интерфейс Comparator содержит ряд методов, ключевым из которых является метод compare():

Листинг 10.1 – Пример интерфейса Коимпаратор

```
public interface Comparator<E> {
    int compare(T a, T b);
    // остальные методы
}
```

Метод compare также возвращает числовое значение - если оно отрицательное, то объект a предшествует объекту b, иначе - наоборот. А если метод возвращает ноль, то объекты равны. Для применения интерфейса нам вначале надо создать класс компаратора, который реализует этот интерфейс:

Листинг 10.2 – Пример класса, реализующего интерфейс Компаратор

```
class PersonComparator implements Comparator<Person>{

    public int compare(Person a, Person b){

        return a.getName().compareTo(b.getName());
```

```
        }
    }
```

Замечание. Предполагаем, что тип Person у нас описан был ранее как класс

Листинг 10.3 – Пример класса Person

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

Сортировка по нескольким критериям

Начиная с JDK 8 в механизм работы компараторов были внесены некоторые дополнения. В частности, теперь мы можем применять сразу несколько компараторов по принципу приоритета. Например, изменим класс Person следующим образом:

Листинг 10.4 – Пример класс Student

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

Здесь добавлено поле для хранения возраста пользователя. И, допустим, нам надо отсортировать пользователей по имени и по возрасту. Для этого определим два компаратора:

Листинг 10.5 – Пример реализующего компаратор

```
class PersonNameComparator implements
Comparator<Person>{
        public int compare(Person a, Person b){
            return a.getName().compareTo(b.getName());
        }
    }
```

Листинг 10.6 – Пример реализующего компаратор

```
    class PersonAgeComparator implements
Comparator<Person>{
        public int compare(Person a, Person b){

            if(a.getAge()> b.getAge())
                return 1;
            else if(a.getAge()< b.getAge())
                return -1;
            else
                return 0;
        }
    }
```

Интерфейс компаратора определяет специальный метод по умолчанию thenComparing, который позволяет использовать цепочки компараторов для сортировки набора:

Листинг 10.7 – Пример реализующего компаратор

```
    Comparator<Person> pcomp = new
PersonNameComparator().thenComparing(new
PersonAgeComparator());

    TreeSet<Person> people = new TreeSet(pcomp);
    people.add(new Person("Tom", 23));
    people.add(new Person("Nick",34));
    people.add(new Person("Tom",10));
    people.add(new Person("Bill",14));

    for(Person  p : people){

        System.out.println(p.getName() + " " +
p.getAge());
    }
```

Задания на практическую работу № 10

Задание 1. (5%)

Создать свой класс Student со всеми переменными экземпляра, конструктором, включающим все переменные, предпочтительно использовать

геттеры и сеттеры для каждой переменной. Класс студент имеет свойства: Имя, Фамилия, Специальность, Курс, Группа

Задание 2. (45%)

Напишите класс SortingStudentsByGPA (может у вас называться Tester или Main, так как содержит функцию main()) создайте поле как массив объектов Student с названием iDNumber, вы можете использовать как массив, так и и ArrayList или TreeSet для хранения данных о студентах

Добавьте методы класса: 1) заполнения массива setArray() 2) метод для сортировки по среднему баллу студентов quicksort() который реализует интерфейс Comparator таким образом, чтобы он сортировал студентов с их итоговым баллом в порядке убывания. В качестве алгоритма сортировки использовать методы сортировок: слиянием и быструю сортировку (добавьте в класс еще один метод). 3)метод для вывода массива студентов outArray()

4)Добавьте в класс возможность сортировать список студентов по другому полю

Задание 3. (50%)

Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках.

Практическая работа № 11. Работа с датой и временем

Цель данной практической работы –научиться работать с датами и временем, применять методы класса Date и Calenadar, других классов для обработки строк.

Теоретические сведения

В Java есть много классов, доступных для работы с датой/временем.

Класс Date

Класс Date изначально предоставлял набор функций для работы с датой - для получения текущего года, месяца и т.д. однако сейчас все эти методы не рекомендованы к использованию и практически всю функциональность для этого предоставляет класс Calendar. Класс Date так же определен в пакете java.sql поэтому желательно указывать полностью квалифицированное имя класса Date.

Существует несколько конструкторов класса Date однако рекомендовано к использованию два

Date() и Date(long date) второй конструктор использует в качестве параметра значение типа long который указывает на количество миллисекунд прошедшее с 1 Января 1970, 00:00:00 по Гринвичу. Первый конструктор создает дату использует текущее время и дату (т.е. время выполнения конструктора). Фактически это эквивалентно второму варианту new Date(System.currentTimeMillis); Можно уже после создания экземпляра класса Date использовать метод setTime(long time), для того, что бы задать текущее время.

Для сравнения дат служат методы after(Date date), before(Date date) которые возвращают булевское значение в зависимости от того выполнено условие или нет. Метод compareTo(Date anotherDate) возвращает значение типа int которое равно -1 если дата меньше сравниваемой, 1 если больше и 0 если даты равны. Метод toString() представляет строковое представление даты, однако для форматирования даты в виде строк рекомендуется пользоваться классом SimpleDateFormat определенном в пакте java.text

Классы Calendar и GregorianCalendar

Более развитые средства для работы с датами представляет класс Calendar. Calendar является абстрактным классом. Для различных платформ реализуются конкретные подклассы календаря. На данный момент существует реализация Грегорианского календаря - GregorianCalendar. Экземпляр этого класса получается вызовом статического метода getInstance(), который возвращает

экземпляр класса Gregorian. Подклассы класса Calendar должны интерпретировать объект Date по-разному. В будущем предполагается реализовать так же лунный календарь, используемый в некоторых странах. Calendar обеспечивает набор методов позволяющих манипулировать различными "частями" даты, т.е. получать и устанавливать дни, месяцы, недели и т.д. Если при задании параметров календаря упущены некоторые параметры, то для них будут использованы значения по умолчанию для начала отсчета. т.е.YEAR = 1970, MONTH = JANUARY, DATE = 1 и т.д.

Для считывания, установки манипуляции различных "частей" даты используются методы get(int filed), set(int filed, int value), add(int field, int amount), roll(int field, inr amount), переменная типа int с именем filed указывает на номер поля с которым нужно произвести операцию. Для удобства все эти поля определены в Calendar, как статические константы типа int. Рассмотрим подробнее порядок выполнения перечисленных методов.

Метод set(int field,int value)

Как уже отмечалось ранее данный метод производит установку какого - либо поля даты. На самом деле после вызова этого метода, немедленного пересчета даты не производится. Пересчет даты будет осуществлен только после вызова методов get(), getTime() или TimeInMillis(). Т.о. последовательная установка нескольких полей, не вызовет не нужных вычислений. Помимо этого, появляется еще один интересный эффект. Рассмотрим следующий пример. Предположим, что дата установлена на последний день августа. Необходимо перевести ее на последний день сентября. Если внутреннее представление даты изменялось бы после вызова метода set, то при последовательной установке полей мы получили бы вот такой эффект.

Листинг 11.1 Пример работы с датой и временем

```
public class Test {
public Test() {
}
public static void main(String[] args) {
SimpleDateFormat sdf = new SimpleDateFormat("yyyy
MMMM dd HH:mm:ss");
Calendar cal = Calendar.getInstance();
cal.set(Calendar.YEAR,2002);
cal.set(Calendar.MONTH,Calendar.AUGUST);
cal.set(Calendar.DAY_OF_MONTH,31);
```

```
    System.out.println(" Initialy set date:”+
sdf.format(cal.getTime()));
    cal.set(Calendar.MONTH,Calendar.SEPTEMBER);
    System.out.println(" Date with month changed :"+
sdf.format(cal.getTime()));
    cal.set(Calendar.DAY_OF_MONTH,30);
    System.out.println(" Date with day changed:”+
sdf.format(cal.getTime()));

    }}
```

Вывод программы листинга 11.1:

```
Initialy set date: 2002 August 31 22:57:47
Date with month changed: 2002 October 01 22:57:47
Date with day changed: 2002 October 30 22:57:47
```

Рассмотрим еще пример.

Листинг 11.2 – Пример работы с датой и временем

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTest {
public static void main(String[] args) {
Date now = new Date();
System.out.println("toString(): " + now);
// dow mon dd hh:mm:ss zzz yyyy
/* SimpleDateFormat может использоваться для
управления форматом отображения даты/времени:
   E (день недели): 3E or fewer (в текстовом формате
xxx), >3E (в полном текстовом формате)
    M (месяц): M (in number), MM ( в числовом виде,
впереди ноль)
    3M: (в текстовом формате xxx), >3M: (в полном
текстовом формате)*/
    //   h (часы): h, hh (with leading zero)
    //   m (минуты)
    //   s (секунды)
    //   a (AM/PM)
    //   H (часы 0 до 23)
    //   z (временная зона)
```

```
    SimpleDateFormat dateFormatter = new
SimpleDateFormat("E, y-M-d 'at' h:m:s a z");
          System.out.println("Format 1:   " +
dateFormatter.format(now));

          dateFormatter = new SimpleDateFormat("E
yyyy.MM.dd 'at' hh:mm:ss a zzz");
          System.out.println("Format 2:   " +
dateFormatter.format(now));
    dateFormatter = new SimpleDateFormat("EEEE, MMMM d,
yyyy");
          System.out.println("Format 3:   " +
dateFormatter.format(now));
```

Вывод программы листинга 11.2

```
toString(): Sat Sep 25 21:27:01 SGT 2010
Format 1:   Sat, 10-9-25 at 9:27:1 PM SGT
Format 2:   Sat 2010.09.25 at 09:27:01 PM SGT
Format 3:   Saturday, September 25, 2010
```

Класса Date будет достаточно, если вам просто нужна простая отметка времени. Вы можете использовать SimpleDateFormat для управления форматом отображения даты /времени. Используйте класс java.util.Calendar, если вам нужно извлечь год, месяц, день, час, минуту и секунду или манипулировать этими полями (например, 7 дней спустя, 3 недели назад).

Используйте java.text.DateFormat для форматирования даты (от даты до текста) и разбора строки даты (от текста к дате). SimpleDateForamt является подклассом DateFormat.

`Date` является устаревшим классом, который не поддерживает интернационализацию. Calendar и DateFormat поддерживают локалзацию (вам нужно учитывать локализацию только в том случае, если ваша программа будет работать во нескольких странах одновременно).

Классы java.util.Calendar и java.util.GregorianCalendar

Рассмотрим пример программы, где мы можем получить год, месяц, день, часы, минуты, секунды:

Листинг 11.3 – Пример использования класса Calendar

```
import java.util.Calendar;
```

```
    public class GetYMDHMS {
    public static void main(String[] args) {
    Calendar cal = Calendar.getInstance();
    // You cannot use Date class to extract individual
Date fields
    int year = cal.get(Calendar.YEAR);
    int month = cal.get(Calendar.MONTH);  // 0 to 11
    int day = cal.get(Calendar.DAY_OF_MONTH);
    int hour = cal.get(Calendar.HOUR_OF_DAY);
    int minute = cal.get(Calendar.MINUTE);
    int second = cal.get(Calendar.SECOND);
    // Заполняем нулями

    System.out.printf("Now is %4d/%02d/%02d
%02d:%02d:%02d\n", year, month+1, day, hour, minute,
second);
       }
    }
```

Измерение времени

Любые приложения (такие как игры и анимация) требуют хорошего контроля времени. Java предоставляет эти статические методы в классе System. Метод System.currentTimeMillis() возвращает текущее время в миллисекундах с 1 января 1970 г. 00:00:00 по Гринвичу (известное как «эпоха») в длинном формате.

Измерение прошедшего времени производится как в примере ниже:

```
    long startTime = System.currentTimeMillis();
    // измерение выполнения кода
      .......
    long estimatedTime = System.currentTimeMillis() -
startTime;
```

Метод System.nanoTime (): возвращает текущее значение наиболее точного доступного системного таймера, в наносекундах, в течение длительного времени. Введенный с JDK 1.5. метод nanoTime () предназначен для измерения относительного временного интервала вместо предоставления абсолютного времени.

Задания на практическую работу № 11

Задание 1. (20%)

Написать программу, выводящую фамилию разработчика, дату и время получения задания, а также дату и время сдачи задания. Для получения последней даты и времени использовать класс Date из пакета java.util.* (Объявление Dated=newDate() или метод System.currentTimeMillis().

Задание 2. (20%)

Приложение, сравнивающее текущую дату и дату, введенную пользователем с текущим системным временем

Задание 3. (20%)

Доработайте класс Student предусмотрите поле для хранения даты рождения, перепишите метод toString() таким образом, чтобы он разработайте метод, возвращал строковое представление даты рождения по вводимому в метод формату даты (например, короткий, средний и полный формат даты).

Задание 4. (10%)

Напишите пользовательский код, который формирует объекты Date и Calendar по следующим данным, вводимым пользователем:

<Год><Месяц><Число>

<Часы1><минуты>

Задание 5 (30%)

Сравнить время выполнения кода в реализации кода в виде различных структур данных из предыдущих заданий (сравнить ArrayList и LinkedList по производительности – операции вставки, удаления, добавления и поиска по образцу)

Практическая работа № 12. Создание программ с графическим интерфейсом пользователя на языке Джава. Компоновка объектов с помощью Layout менеджеров

Цель: цель данной практической работы - научиться создавать графический интерфейс пользователя, освоить на практике работу с различными объектами для создания GUI, менеджерами размещения компонентов.

Теоретические сведения

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

КомпонентTextField

Компонент 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 создания 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 методаgetText(). 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.

Листинг 12.1 – Пример пользовательского класса окна, наследуемся от JFrame

```
    class LabExample extends JFrame{
    JTextField jta = new JTextField(10);.
    //можно задать свойства шрифта
    Font fnt = new Font("Times new roman",Font.BOLD,20);
LabExample(){
    // вызываем конструктор родителя JFrame
    super("Example");
    //устанавливаем менеджер компоновки FlowLayout
    setLayout(new FlowLayout());
    //устанавливаем размеры окна
    setSize(250,100);
```

```
//добавляем текстовое поле к окну
add(jta);
//задаем цвет фона
jta.setForeground(Color.PINK);
//задаем цвет шрифта
jta.setFont(fnt);
setVisible(true);
}
public static void main(String[]args){
//вызываем конструктор класса LabExample
new LabExample();
}
}
```

На рис.12.1 представлен пример работы программы на листинге 12.1.

Рисунок 12.1. Пример работы программы на листинге 12.1

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:

Менеджеры компоновки: BorderLayout

Разделяет компонент на пять областей (WEST, EAST, NOTH, SOUTH and Center). Другие компоненты могут быть добавлены в любой из этих компонентов пятерками.

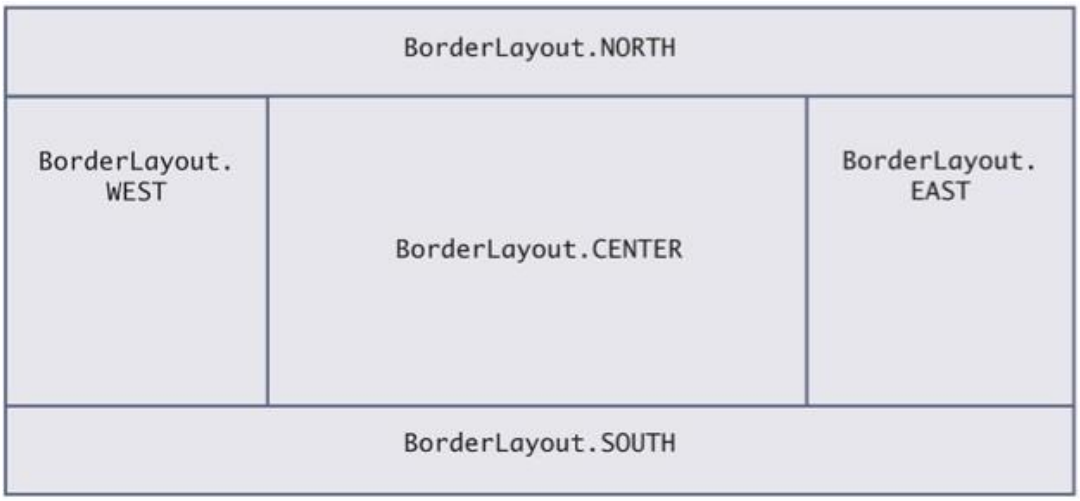

Рисунок 12.2. Схема менеджера BorderLayout

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 вэти 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	1	1	1
	0	1	2

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

Следующий пример иллюстрирует смесь компоновки различных компонентов

Листинг 12.2 – Пример программы

```
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(newJButton("one"));
pnl[10].add(new JButton("two"));
pnl[10].add(new JButton("three"));
pnl[10].add(new JButton("four"));
pnl[10].add(new JButton("five"));
setSize(800,500);
}
public static void main(String[]args){
new BorderExample().setVisible(true);
}
}
```

Вот такой будет иметь вид, представленный выше код

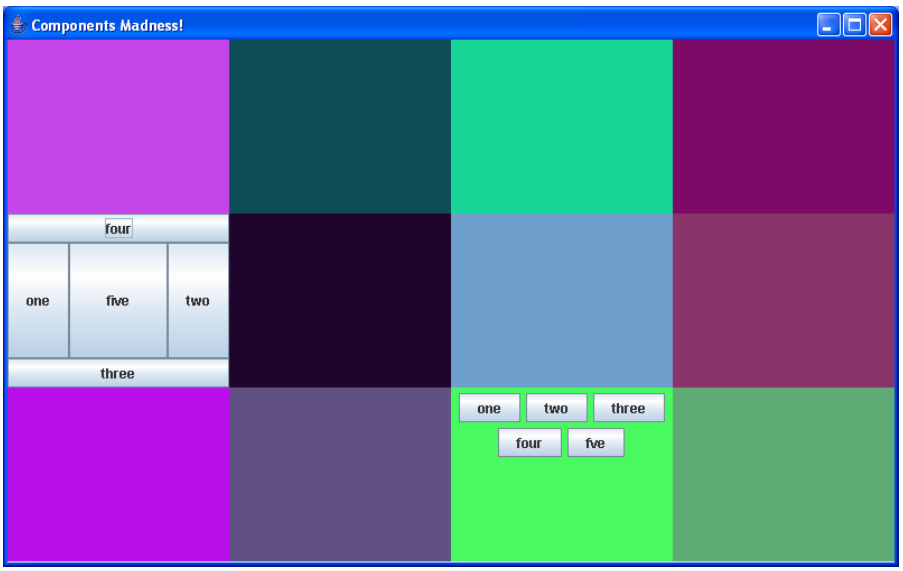

Рисунок 12.3. Пример работы программы на листинге 11.4

Заметьте, что компонент JFrame использует менеджер компоновки GridLayout в виде таблицы размера 3 на 4, в то время как компонент JPanel размером 2 на 1 использует менеджер компоновки BorderLayout. А компонент JPanel размер 3 на 3 использует менеджер компоновки FLowLayout.

Менеджер Layout по умолчанию

Иногда бывает нужно изменить размер и расположение компонента в контейнере. Таким образом, мы должны указать программе не использовать никакой менеджер компоновки, то есть использовать setLayout (null). Так что мы получим что-то вроде этого:

Листинг 12.3 – Пример программы

```
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);
// добаволяем 150,300 ширина = 100, высота =20
but2.setSize(80,400);
// добавляем к 0,0 ширина = 80, высота =400
but3.setLocation(300,1 00);
but3.setSize(200,75);
```

```
    // those two steps can be combined in onesetBounds
method call
    add(but1);
    add(but2);
    add(but3);
    setSize(500,500);
    }
    public static void main(String[]args){
    new NullLayout().setVisible(true);
     }
    }
```

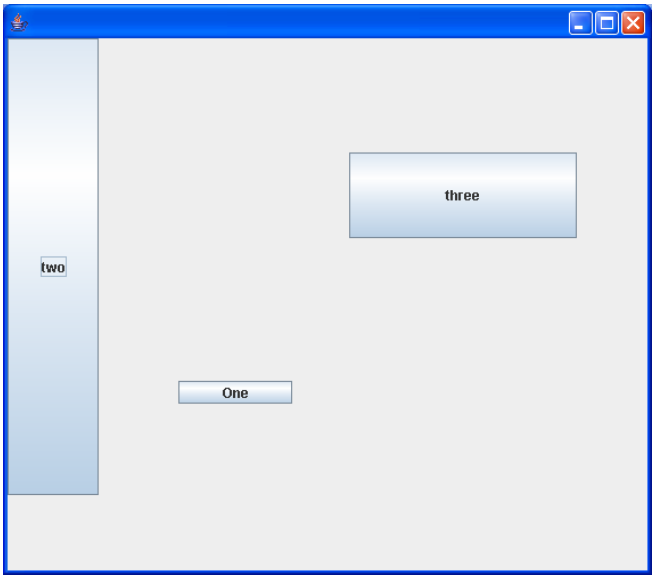

Рисунок 12.4. Пример работы программы на листинге 12.3

Организация меню

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

JMenuBar: который представляет собой компонент, который содержит меню.

JMenu: который представляет меню элементов для выбора.

Рисунок 12.5. Диаграмма класса MovableRectangle.

JMenuItem: представляет собой элемент, который можно кликнуть из меню.

Подобно компоненту Button (на самом деле MenuItems являются

подклассами класса AbstractButton). Мы можем добавить ActionListenerк ним так же, как мы делали с кнопками

Задания на практическую работу № 12

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.

Практическая работа № 13. Обработка строк в Java

Цель: закрепить знания в области обработки строк, научиться применять методы класса String и других классов для обработки строк.

Теоретические сведения

Для работы с текстовыми данными в Джава есть три класса: String, StringBuffer и StringBuilder.

Особенности использования строк в Джава

В Джава строки представляют собой неизменяемую последовательность символов Unicode. В отличие от представления в C / C ++, где строка является просто массивом типа char, любая Java, строка является объектом класса java.lang.

Однако Джава строка, представляет собой в отличие от других используемых классов особый класс, который обладает довольно специфичными характеристиками. Отличия класса строк от обычных классов:

- строка в Джава представляет из себя строку литералов (текст), помещенных в двойные кавычки, например:

"Hello, World! ". Вы можете присвоить последовательность строковых литералов непосредственно переменной типа String, вместо того чтобы вызывать конструктор для создания экземпляра класса String.

- Оператор '+' является перегруженным, для объектов типа String, и всегда используется, чтобы объединить две строки операндов. В данном контексте мы говорим об операции конкатенации или сложения строк. Хотя '+' не работает как оператор сложения для любых других объектов, кроме строк, например, таких как Point и Circle.
- Строка является неизменяемой, то есть, символьной константой. Это значит, что ее содержание не может быть изменено после ее (строки как объекта) создания. Например, метод toUpperCase () – преобразования к верхнему регистру создает и возвращает новую строку вместо изменения содержания существующей строки.

Обратитесь к API JDK для того чтобы ознакомиться с полным списком возможностей класса String в java.lang.String. Наиболее часто используемые методы класса String приведены ниже.

```
int length()        // возвращает длину String
boolean isEmpty()// то же что thisString.length == 0
// сравнение
```

```
    boolean equals(String another)
    // НЕЛЬЗЯ использовать'==' или'!=' для сравнения
объектов String в Java
    boolean equalsIgnoreCase(String another)
    int compareTo(String another)// возвращает 0 если эта
строка совпадает с another;
    // <0 если лексикографически меньше another; or >0
    int compareToIgnoreCase(String another)
    boolean startsWith(String another)
    boolean startsWith(String another, int fromIndex)  //
поиск начинается с fromIndex
    boolean endsWith(String another)
    // поиск & индексирование
    int indexOf(String search)
    int indexOf(String search, int fromIndex)
    int indexOf(int character)
    int indexOf(int character, int fromIndex)     // поиск
вперед от fromIndex
    int lastIndexOf(String search)
    int lastIndexOf(String search, int fromIndex)   //
поиск назад от fromIndex
    int lastIndexOf(int character)
    int lastIndexOf(int character, int fromIndex)
    // выделение char или части строки из String
(подстрока)
    char charAt(int index)              // позиция от 0 до
(длина строки-1)
    String substring(int fromIndex)
    String substring(int fromIndex, int endIndex)   //
exclude endIndex
    // создается новый String или char[] из исходного
(Strings не изменяются!)
    String toLowerCase()
    //преобразование к нижнему регистру
    String toUpperCase()
    //преобразование к верхнему регистру
    String trim()
    // создается новый String с помощью удаления пробелов
```

```
спереди и сзади
    String replace(char oldChar, char newChar)
    //создание нового String со старым oldChar
перемещается посредством буфера newChar
    String concat(String another)
    // то же самое какthisString + другое
    char[] toCharArray()
    // создается char[] из string
    void getChars(int srcBegin, int srcEnd, char[] dst,
int dstBegin)
    // копируется в массив назначения dst char[]
    /* статические методы для преобразования примитивов в
String*/
    static String ValueOf(type arg)
     // тип может быть примитивный или char[]
    // статические методы дают форматированный String
используя //спецификаторы форматирования
    static String format(String formattingString,
Object... args)
    // так же как printf()
    // регулярные выражения (JDK 1.4)
    boolean matches(String regexe)
    String replaceAll(String regexe, String replacement)
    String replaceAll(String regexe, String replacement)
    String[] split(String regexe)
    // разделяет String используя regexe как разделитель,
    // возвращает массив String
    String[] split(String regexe, int count)
    // для подсчета количества раз только (count)
```

Статический метод String.format()

Статический метод String.format() (введен в JDK 1.5) может быть использован для получения форматированного вывода, таким же образом как это делается в языке Си с использованием функции printf() и спецификаторов вывода для различных типов данных. Метод format() делает то же самое, что функция printf(). Например:

```
String.format("%.1f", 1.234);
// возвращает String "1.2"
```

Удобно использовать String.format(), если вам нужно получить простую отформатированную строку для некоторых целей (например, для использования в методе ToString()). Для сложных строк, нужно использовать StringBuffer/StringBuilder с Formatter. Если вам просто нужно отправить простую отформатированную строку на консоль, то просто воспользуйтесь методом System.out.printf(), например:

```
System.out.printf("%.1f", 1.234);
```

Пример использования методов lastIndexOf() и substring() в пользовательском классе Filename

Листинг программы 13.1 – Пример класс Filename

```
public class Filename {
private String fullPath;
private char pathSeparator, extensionSeparator;
public Filename(String str, char sep, char ext) {
fullPath = str;
pathSeparator = sep;
extensionSeparator = ext;
}
public String extension() {
int dot = fullPath.lastIndexOf(extensionSeparator);
return fullPath.substring(dot + 1);
}
// получение имени файла без расширения
public String filename() {
int dot = fullPath.lastIndexOf(extensionSeparator);
int sep = fullPath.lastIndexOf(pathSeparator);
return fullPath.substring(sep + 1, dot);
 }
public String path() {
int sep = fullPath.lastIndexOf(pathSeparator);
return fullPath.substring(0, sep);
}
```

Теперь рассмотрим программу Main, которая использует класс Filename:

Листинг программы 13.2 – Пример программы обрабатывающей строки

```
public class Main{
 public static void main(String[] args) {
  final String FPATH = "/home/user/index.html";
  Filename myHomePage = new Filename(FPATH, '/', '.');
```

```
        System.out.println("Extension = " +
myHomePage.extension());
      System.out.println("Filename = " +
myHomePage.filename());
      System.out.println("Path = " + myHomePage.path());
         }
     }
```

Результат работы программы:

```
Extension = html
Filename = index
Path = /home/user
```

Особенности класса String

Строки получили специальное значение в Java, потому что они часто используются в любой программе. Следовательно, эффективность работы с ними (с точки зрения вычислений и хранения) имеет решающее значение. Что нужно помнить про класс **String**

- это immutable (неизменный) класс
- это final класс

Разработчики Джава все-таки решили сохранить примитивные типы в объектно-ориентированном языке вместо того, чтобы сделать вообще все в виде объектов. Нужно сказать, что сделано это в первую очередь, для того чтобы повысить производительность языка. Ведь примитивы хранятся в стеке вызовов, и следовательно, требуют меньше пространства для хранения, и ими легче управлять. С другой стороны, объекты хранятся в области памяти, которую используют программы, и которая называется “куча” (heap), а этот механизм требуют сложного управления памятью и потребляет гораздо больше места для хранения.

По соображениям производительности, класс String в Джаваразработан, так, чтобы быть чем-то промежуточным между примитивными типами данных и типами данных типа класс. Как уже было отмечено выше специальные характеристики типа String включают в себя:

- '+' оператор, который выполняет сложение примитивных типов данных (таких, как int и double), и перегружен, чтобы работать на объектах String. Операция '+’ выполняет конкатенацию двух операндов типа String.
- Джава не поддерживает механизма перегрузки операций по разработке программного обеспечения. В языке, который поддерживает перегрузку операций, например C ++, вы можете превратить оператор '+' (с помощью перегрузки) в оператор для выполнения сложения или вообще

вычитания, например двух матриц, кстати это будет примером плохого кода. В Джава оператор '+' является единственным оператором, который внутренне перегружен, чтобы поддержать конкатенацию (сложение) строк в Java. Нужно принять к сведению, что '+' не работает на любых других произвольных объектах, помимо строк, например, таких как рассмотренные нами ранее классы Point или Circle.

Теперь, собственно, о строках. Существуют несколько способов создания строк. Строка String может быть получена одним из способов:

- непосредственно из присвоения строкового литерала ссылке типа String – таким же способом как примитивные типы данных;
- или с помощью оператора new и конструктора класса String, аналогично вызову конструктора любого другого класса. Тем не менее, этот способ не часто используется и использовать его не рекомендуется.

Для примера:

```
String str1 = "Java is Hot";
// неявный вызов конструктора через присваивание строкового литерала
String str2 = new String("I'm cool");
// явный вызов конструктора через new
char[] array = { 'h', 'e', 'l', 'l', 'o', '.' };
//еще один способ создания строки
```

В первом случае str1 объявлена как ссылка типа String и инициализируется строкой “Java is Hot”. Во второй строке, str2 объявлена как ссылка на строку и инициализируется с помощью вызова оператора new и конструктора, который инициализирует ее значением “I'm cool”. Строковые литералы хранятся в общем пуле. Это облегчает совместное использование памяти для строк с тем же содержанием в целях сохранения памяти. Объекты строк, выделенные с помощью оператора new, хранятся в куче (heap), а там нет разделяемого хранилища для того же самого контента (содержания).

Строковые литералы и объекты типа String

Как уже упоминалось, есть два способа создания строк: неявное создание путем присвоения строкового литерала переменной или явного создания объекта String, через вызов оператора new и вызов конструктора. У класса String есть еще одна особенность. Все строковые литералы, определенные в Джава коде, вроде "asdf", на этапе компиляции кэшируются и добавляются в так называемый пул строк. Например:

```
String s1 = "Hello";                // String литерал
```

```
String s2 = "Hello";                 // String литерал
String s3 = s1;                     // одинаковые ссылки
String s4 = new String("Hello");   // String объект
String s5 = new String("Hello");   // String объект
```

Джава предоставляет программистам специальный механизм для хранения последовательностей символьных литералов (строк), так называемый общий пул строк. Если две последовательности литералов (строки) имеют одинаковое содержание, то они разделяют общее пространство для хранения внутри общего пула. Такой подход принят для того, чтобы сохранить место для хранения часто используемых строк. С другой стороны, объекты типа String (строки), созданные с помощью оператора new и конструктора хранятся в куче. Механизм работы со строками в Джава представлен на рис.13.1.

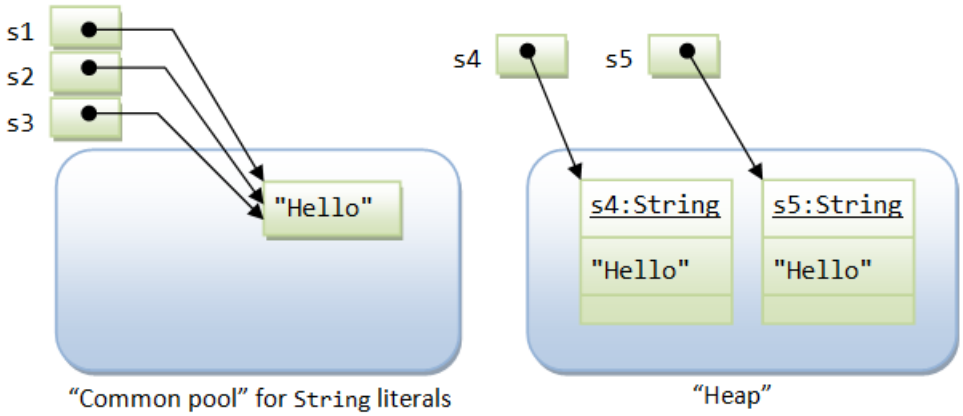

Рисунок 13.1 Механизм работы со строками в Джава

Каждый объект String в куче имеет свое собственное место для хранения, как и любой другой объект. Там нет обмена хранения в куче, даже если два объекта Строковые имеют, то же содержание.

А в куче нет разделяемого пространства для хранения двух объектов, даже если эти два объекта являются объектами типа String и имеют одинаковое содержание.

Вы можете использовать метод equals() класса String для сравнения содержимого двух строк. Вы можете использовать оператор сравнения на равенство '==', чтобы сравнить ссылки (или указатели) двух объектов. Изучите следующие программные примеры:

```
s1 == s1;  // true, одинаковые ссылки
s1 == s2;  // true, s1 and s1 разделяют общий пул
s1 == s3;  /* true, s3 получает то же самое значение
что ссылка s1*/
s1.equals(s3);  // true, одинаковое содержимое
s1 == s4;   // false, различные ссылки
```

```
s1.equals(s4);  // true, одинаковое содержимое
s4 == s5;  // false, различные ссылки в куче
s4.equals(s5);  // true, одинаковое содержимое
```

Важные замечания

- В приведенном выше примере, используется оператор отношения для того, чтобы проверить на равенство ‘==‘ ссылки двух объектов String. Это сделано, чтобы показать различия между строковыми последовательностями литералов, которые используют совместное пространство для хранения в общем пуле строк и объектов String, созданных в куче. Это логическая ошибка в использовании выражения (str1 == str2) в программе, чтобы сравнить содержимое двух объектов типа String.
- Строка может быть создана непосредственно путем присваивания последовательности литералов (строки), которая разделяет общий пул строк. Не рекомендуется использовать оператор new для создания объектов String в куче.

Помните о том: что строки в Джава являются неизменяемыми!

С тех самых пор, когда в языке Джава появились возможности по использованию разделяемого пространства для хранения строк с одинаковым содержанием в виде строкового пула, String в Джава стали неизменяемыми. То есть, как только строка создается (как объект в памяти программы), ее содержание не может быть изменено никаким образом (по аналогии с Си – строковые литералы — это символьные константы). В противном случае, если этого не сделать, другие ссылки String разделяющие ту же самую ячейку памяти будут зависеть от изменений, которые могут быть непредсказуемыми и, следовательно, является нежелательными. Такой метод, как например, toUpperCase () казалось-бы может изменить содержимое объекта String. Хотя на самом деле, создается совершенно новый объект String и возвращается как раз он в точку вызова. Исходный объект-строка будет впоследствии удален сборщиком мусора (Garbage-collected), как только не окажется больше ссылок, которые ссылаются на него.

Вот поэтому-то объект типа String и считается неизменяемым объектом, вследствие этого, считается не эффективным использовать тип String, например, в том случае, если вам нужно часто модифицировать строку (так вы в таком случае будете создавать много новых объектов типа String, которые каждый раз будут занимать новые места для хранения). Например,

```
// неэффективный код
String str = "Hello";
for (int i = 1; i < 1000; ++i) {
```

```
        str = str + i;
    }
```

Выводы.

Если содержимое строки должно часто меняться в вашей программе, используйте классы StringBuffer или StringBuilder вместо класса String.

Классы StringBuffer и StringBuilder

Классы StringBuffer и StringBuilder в Джаваиспользуются, когда возникает необходимость сделать много изменений в строке символов.В отличие от String, объекты типа StringBuffer и StringBuilder могут быть изменены снова и снова. В ДжаваStringBuilder был введен начиная с Джава5.

Как объяснялось выше, строки String являются неизменяемыми, поэтому строковые литералы с таким контентом хранятся в пуле строк. Изменение содержимого одной строки непосредственно может вызвать нежелательные побочные эффекты и может повлиять на другие строки, использующие ту же память.

JDK предоставляет два класса для поддержки возможностей по изменению строк: это классы StringBuffer и StringBuilder (входят в основной пакет java.lang).

Объекты StringBuffer или StringBuilder так же, как и любые другие обычные объекты, которые хранятся в куче, а не совместно в общем пуле, и, следовательно, могут быть изменены, не вызывая нехороших побочных эффектов на другие объекты.

Класс StringBuilder как класс был введён в JDK 1.5. Это то же самое, как использование класса StringBuffer, за исключением того, что StringBuilder не синхронизирован по многопоточным операциям. Тем не менее, для программы в виде одного потока или нити управления, использование класса StringBuilder, без накладных расходов на синхронизацию, является более эффективным.

Использование StringBuffer

Класс находится в пакете **java.lang**. Прочитайте спецификацию API JDK для использования java.lang.StringBuffer.

Методы класса:

```
// конструкторы
StringBuffer()                  // инициализация пустым
anStringBuffer
StringBuffer(int size)        // определяет размер при
инициализации
```

```
    StringBuffer(String   s)              //инициализируетя
содержимым   s
    // длина строки
    int length()
    // Методы для конструирования содержимого
    StringBuffer append(type arg)   // тип может быть
примитивным, char[], String, StringBuffer, и т.д.
    StringBuffer insert(int offset, arg)

    // Методы для манипуляции содержимым
    StringBuffer delete(int start, int end)
    StringBuffer deleteCharAt(int index)
    void setLength(int newSize)
    void setCharAt(int index, char newChar)
    StringBuffer replace(int start, int end, String s)
    StringBuffer reverse()

    // Методы для выделения целого/части содержимого
    char charAt(int index)
    String substring(int start)
    String substring(int start, int end)
    String toString()
    // Методы для поиска
    int indexOf(String searchKey)
    int indexOf(String searchKey, int fromIndex)
    int lastIndexOf(String searchKey)

    int lastIndexOf(String searchKey, int fromIndex)
```

Обратите внимание, что объект класса StringBuffer является обычным объектом в прямом понимании этого слова. Вам нужно будет использовать конструктор для создания объектов типа класс StringBuffer (вместо назначения в строку буквальном). Кроме того, оператор '+' не применяется к объектам, в том числе и к объектам StringBuffer. Вы должны будете использовать такой метод, как append() или insert() чтобы манипулировать StringBuffer.

Чтобы создать строку из частей, более эффективно использовать класс StringBuffer (для многопоточных программ) или StringBuilder (для однопоточных), вместо конкатенации строк. Например

Листинг 13.3 – Пример создания строки с помощью StringBuilder и String

```
    // Создадим строку типа YYYY-MM-DD HH:MM:SS
    int year = 2010, month = 10, day = 10;
    int hour = 10, minute = 10, second = 10;
    String dateStr = new
StringBuilder().append(year).append("").append(month).app
end("").append(day).append("").append(hour)
    .apend(":").append(minute).append(":").append(second)
.toString();
    System.out.println(dateStr);

    // StringBuilder более эффективная конкатенация строк
    String anotherDataStr = year + "-" + month + "-" + day
+ " " + hour + ":" + minute + ":" + second;
    System.out.println(anotherDataStr);
```

Компилятор JDK, по сути, использует оба класса как String, так и StringBuffer для обработки конкатенации через операцию сложения строк '+'.

Для примера, рассмотрим строчку кода:

```
String msg = "a" + "b" + "c";
```

Представленная выше строчка кода будет скомпилирован в следующую для повышения эффективности:

```
String msg = new
StringBuffer().append("a").append("b").append("c").toStri
ng();
```

В этом процессе создаются промежуточный объект StringBuffer и возвращаемый объект String.

Использование класса StringBuilder

Класс находится в пакете **java.lang**. Прочитайте спецификацию API JDK для использования. java.lang.StringBuilder

Программа ниже демонстрирует разные способы инвертирования длинных строк. Сравниваются три способа работы со строками: как с объектами класса String, так и с помощью StringBuffer и StringBuilder с использованием метода reverse(). Для измерения времени выполнения различных участков кода в примере используется метод

public static native long nanoTime();

Этот метод возвращает текущее значение наиболее точное время системных часов (таймера), в наносекундах. Его можно использовать только для

измерения затраченного времени на выполнение операций, и он никак не связан с системным временем и текущим мировым временем.

Возвращаемое методом значение представлено в виде наносекунд, с момента фиксации, на любой произвольный момент времени. Например, нам необходимо определить, сколько времени занимает выполнение некоторого кода, для этого необходимо выполнить следующее (см Листинг 13.4):

Листинг 13.4– Пример программы

```
    /*
    long startTime = System.nanoTime();
    // ... the code being measured ...
    long estimatedTime = System.nanoTime() - startTime;
    * @return The current value of the system timer, in
nanoseconds.
    * @since 1.5
    */
// Риверс длинной строки с помощью String и StringBuffer
    public class StringsBenchMark {
    public static void main(String[] args) {
    long beginTime, elapsedTime;
    // Build a long string
    String str = "";
    int size = 16536;
    char ch = 'a';
    // Эталонное время в наносекундах
    beginTime = System.nanoTime();
    for (int count = 0; count < size; ++count) {
      str += ch;
      ++ch;
      if (ch > 'z') {
        ch = 'a';
       }
     }
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed    Time    is    "    +
elapsedTime/1000 + " usec (Build String)");
    /* Риверс строки, строим другую строку за символом в
обратном порядке*/
    String strReverse = "";
```

```
    beginTime = System.nanoTime();
    for (int pos = str.length() - 1; pos >= 0 ; pos--) {
    strReverse += str.charAt(pos);   // Конкатенация
    }
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed    Time    is    "    +
elapsedTime/1000 + " usec (Using String to reverse)");

    // Риверс строки через пустой StringBuffer путем
добавления символов в обратном порядке
    beginTime = System.nanoTime();
    StringBuffer sBufferReverse = new StringBuffer(size);
    for (int pos = str.length() - 1; pos >= 0 ; pos--) {
    sBufferReverse.append(str.charAt(pos));     // append
    }
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed    Time    is    "    +
elapsedTime/1000   +   "   usec   (Using   StringBuffer   to
reverse)");

    /* Reverse a String by creating a StringBuffer with
the given String and invoke its reverse()*/
    beginTime = System.nanoTime();
    StringBuffer     sBufferReverseMethod     =     new
StringBuffer(str);
    sBufferReverseMethod.reverse();      // use reverse()
method
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed    Time    is    "    +
elapsedTime/1000 + " usec (Using StringBuffer's reverse()
method)");

    /* Reverse a String via an empty StringBuilder by
appending characters in the reverse order*/
    beginTime = System.nanoTime();
    StringBuilder       sBuilderReverse       =       new
StringBuilder(size);
    for (int pos = str.length() - 1; pos >= 0 ; pos--) {
```

```
    sBuilderReverse.append(str.charAt(pos));
    }
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed Time is " +
elapsedTime/1000 + " usec (Using StringBuilder to
reverse)");

    // Reverse a String by creating a StringBuilder with
the given String and invoke its rever its reverse()*/
    beginTime = System.nanoTime();
    StringBuffer sBuilderReverseMethod = new
StringBuffer(str);
    sBuilderReverseMethod.reverse();
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed Time is " +
elapsedTime/1000 + " usec (Using StringBuidler's reverse()
method)");
       }
    }
```

Задания на практическую работу № 13

Задание 1 на класс String (5%)

1. Напишите метод, который принимает в качестве параметра любую строку, например “I like Java!!!”.
2. Распечатать последний символ строки. Используем метод String.charAt().
3. Проверить, заканчивается ли ваша строка подстрокой “!!!”. Используем метод String.endsWith().
4. Проверить, начинается ли ваша строка подстрокой “I like”. Используем метод String.startsWith().
5. Проверить, содержит ли ваша строка подстроку “Java”. Используем метод String.contains().
6. Найти позицию подстроки “Java” в строке “I like Java!!!”.
7. Заменить все символы “a” на “o”.
8. Преобразуйте строку к верхнему регистру.
9. Преобразуйте строку к нижнему регистру.
10. Вырезать строку Java с помощью метода String.substring().

Задание 2. (5%) Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество. Программу оптимизировать с точки зрения быстродействия.

Задание 3. (15%) Доработать класс адреса, который из полученной строки формата “Страна[d] Регион[d] Город[d] Улица[d] Дом[d] Корпус[d] Квартира” ([d] – разделитель, например, «запятая») выбирает соответствующие части и записывает их в соответствующие поля класса Address. Учесть, что в начале и конце разобранной части адреса не должно быть пробелов. Все поля адреса строковые. Разработать проверочный класс не менее чем на четыре адресных строки. В программе предусмотреть две реализации этого метода:

а) разделитель – только запятая (использовать метод split()); Внимание, при разработке нужно учесть, что

б) разделитель – любой из символов ‘,’ ‘.’ ‘;’ - (класс StringTokenizer[8]).

Задание 4. (25%) Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно.

Дан также строковый массив: shirts[0] = "S001,Black Polo Shirt,Black,XL"; shirts[1] = "S002,Black Polo Shirt,Black,L"; shirts[2] = "S003,Blue Polo Shirt,Blue,XL"; shirts[3] = "S004,Blue Polo Shirt,Blue,M"; shirts[4] = "S005,Tan Polo Shirt,Tan,XL"; shirts[5] = "S006,Black T-Shirt,Black,XL"; shirts[6] = "S007,White T-Shirt,White,XL"; shirts[7] = "S008,White T-Shirt,White,L"; shirts[8] = "S009,Green T-Shirt,Green,S"; shirts[9] = "S010,Orange T-Shirt,Orange,S"; shirts[10] = "S011,Maroon Polo Shirt,Maroon,S";

Преобразуйте строковый массив в массив класса Shirt и выведите его на консоль.

Задание 5. (25%) Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов:

+<Код страны><Номер 10 цифр>, например “+79175655655” или

“+104289652211” или

8<Номер 10 цифр> для России, например “89175655655”

и преобразует полученную строку в формат:

+<Код страны><Три цифры>–<Три цифры>–<Четыре цифры>

Задание 6. (25%) В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя

[8]https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html#:~:text=The%20string%20tokenizer%20class%20allows,they%20recognize%20and%20skip%20comments.

StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

Практическая работа № 14. Использование регулярных выражений в Джава приложениях

Цель: практической работы – понять особенности использования регулярных выражений в Java, научиться работать с строками и применять регулярные выражения для обработки строк в программах.

Теоретические сведения

Регулярные выражения и их использование в Java программах для обработки строк

Регулярные выражения – эта система обработки текста, основанная на специальной системе записи образцов для поиска. Образец (pattern), задающий правило поиска, по-русски также иногда называют «шаблоном», «маской». Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Язык программирования Java также поддерживает регулярные выражения для работы со строками.

Основными классами для работы с регулярные выражения являются класс java.util.regex.Pattern и класс java.util.regex.Matcher.

Для определения шаблона применяются специальные синтаксические конструкции.

Класс Pattern

Класс java.util.regex.Pattern применяется для определения регулярных выражений, для которого ищется соответствие в строке, файле или другом объекте представляющем собой некоторую последовательность символов. Этот класс используется для простой обработки строк. Для более сложной обработки строк используется класс Matcher, рассматриваемый ниже.

В классе Pattern объявлены следующие методы:

- compile(String regex) – возвращает Pattern, который соответствует regex;
- matcher(CharSequence input) – возвращает Matcher, с помощью которого можно находить соответствия в строке input;
- matches(String regex, CharSequence input) – проверяет на соответствие строки input шаблону regex;
- pattern() – возвращает строку, соответствующую шаблону;
- split(CharSequence input) – разбивает строку input, учитывая, что разделителем является шаблон;

- split(CharSequence input, int limit) – разбивает строку input на не более чем limit частей.

С помощью метода matches() класса Pattern можно проверять на соответствие шаблону целой строки, но если необходимо найти соответствия внутри строки, например, определять участки, которые соответствуют шаблону, то класс Pattern не может быть использован. Для таких операций необходимо использовать класс Matcher.

Класс Matcher

С помощью класса java.util.regex.Matcher можно получить больше информации каждом соответствии.

Начальное состояние объекта типа Matcher не определено. Попытка вызвать какой-либо метод класса для извлечения информации о найденном соответствии приведет к возникновению ошибки IllegalStateException. Для того чтобы начать работу с объектом Matcher нужно вызвать один из его методов:

- matches() – проверяет, соответствует ли вся строка шаблону;
- lookingAt() – пытается найти последовательность символов, начинающуюся с начала строки и соответствующую шаблону;
- find() или find(int start) – пытается найти последовательность символов, соответствующих шаблону, в любом месте строки. Параметр start указывает на начальную позицию поиска.

Иногда необходимо сбросить состояние объекта класса Matcher в исходное, для этого применяется метод reset() или reset(CharSequence input), который также устанавливает новую последовательность символов для поиска.

Для замены всех подпоследовательностей символов, удовлетворяющих шаблону, на заданную строку можно применить метод replaceAll(String replacement).

Для того чтобы ограничить поиск границами входной последовательности применяется метод region(int start, int end), а для получения значения этих границ – regionEnd() и regionStart(). С регионами связано несколько методов:

useAnchoringBounds(boolean b) – если установлен в true, то начало и конец региона соответствуют символам ^ и $ соответственно;

hasAnchoringBounds() – проверяет закрепленность границ.

В регулярном выражении для более удобной обработки входной последовательности применяются группы, которые помогают выделить части найденной подпоследовательности. В шаблоне они обозначаются скобками «(» и «)». Номера групп начинаются с единицы. Нулевая группа совпадает со всей

найденной подпоследовательностью. Далее приведены методы для извлечения информации о группах:

end() – возвращает индекс последнего символа подпоследовательности, удовлетворяющей шаблону;

end(int group) – возвращает индекс последнего символа указанной группы;

group() – возвращает всю подпоследовательность, удовлетворяющую шаблону;

group(int group) – возвращает конкретную группу;

groupCount() – возвращает количество групп;

start() – возвращает индекс первого символа подпоследовательности, удовлетворяющей шаблону;

start(int group) – возвращает индекс первого символа указанной группы;

hitEnd() – возвращает истину, если был достигнут конец входной последовательности.

Следующий пример показывает использование возможностей классов Pattern и Matcher, для поиска, разбора и разбиения строк.

Листинг 14.1 – Пример использования классов Pattern и Matcher,

```
import java.util.regex.*;
public class DemoRegular {
public static void main(String[] args) {
// проверка на соответствие строки шаблону
Pattern p1 = Pattern.compile("a*y");
Matcher m1 = p1.matcher( "aaay" );
boolean b = m1.matches();
System.out.println(b);
// поиск и выбор подстроки, заданной шаблоном
String regex = "(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*" ;
String s = "адреса эл.почты: mymail@tut.by и rom@bsu.by";
Pattern p2 = Pattern. compile (regex);
Matcher m2 = p2.matcher(s);
while (m2.find()) {
System.out.println("e-mail: " + m2.group());
```

```
}
// разбиение строки на подстроки с применением шаблона в
качестве
// разделителя
Pattern p3 = Pattern.compile("\\d+\\s?");
String[] words = p3.split("java5tiger 77 java6mustang");
for (String word : words)
System.out.println(word);
}
}}
```

В результате работы программы будет выведено:

```
true
e - mail : mymail @ tut. by
e-mail: rom@bsu.by
java
tiger
java
mustang
```

Следующий пример демонстрирует возможности использования групп, а также собственных и неполных квантификаторов.

Листинг 14.2 – Пример использования квантификаторов

```
import java.util.regex.*;
public class Groups {
public static void main(String[] args) {
String input = "abdcxyz";
myMatches("([a-z]*)([a-z]+)", input);
myMatches("([a-z]?)([a-z]+)", input);
myMatches("([a-z]+)([a-z]*)", input);
```

```
}}
```

В результате работы программы будет выведено:

```
First group: abdcxy
Second group: z
First group: a
Second group: bdcxyz
First group: abdcxyz
Second group: nothing
```

В первом случае к первой группе (First group) относятся все возможные символы, но при этом остается минимальное количество символов для второй группы (Second group). Во втором случае для первой группы выбирается наименьшее количество символов, т. к. используется слабое совпадение. В третьем случае первой группе будет соответствовать вся строка, а для второй не остается ни одного символа, так как вторая группа использует слабое совпадение. В четвертом случае строка не соответствует регулярному выражению, т. к. для двух групп выбирается наименьшее количество символов.

В классе Matcher объявлены два полезных метода для замены найденных подпоследовательностей во входной строке.

Matcher appendReplacement(StringBuffer sb, String replacement) – метод читает символы из входной стоки и добавляет их в sb. Чтение останавливается на start() – первой позиции предыдущего совпадения, после чего происходит добавление в sb строки replacement. При следующем вызове этого метода, производится добавление символов, начиная с символа с индексом end() предыдущего совпадения.

Класс StringTokenizer

Этот класс предназначен для разбора строки по лексемам (tokens). Строка которую необходимо разобрать передается в качестве параметра конструктору StringTokenizer(String str). Определено еще два перегруженных конструктора, которым дополнительно можно передать строку-разделитель лексем StringTokenizer(String str,String delim) и признак возврата разделителя лексем StringTokenizer(String str,String delim,Boolean retirnDelims). Разделителем лексем по умолчанию служит пробел.

Листинг 14.3 – Пример класса `Test`

```
public class Test {

  public Test() {
}
public static void main(String[] args) {
Test test = new Test();
String toParse = “word1;word2;word3;word4”;
StringTokenizer st = new StringTokenizer(toParse,";");
while(st.hasMoreTokens()){
System.out.println(st.nextToken());
}
}
}
```

Вывод:

```
word1
word2
word3
word4
```

Задания на практическую работу № 14

1. Необходимо реализовать консольное приложение, позволяющее манипулировать строкой, разбив ее на элементы путем использования регулярных выражений.
2. Написать регулярное выражение, определяющее является ли данная строка строкой "abcdefghijklmnopqrstuv18340" или нет.
   a) пример правильных выражений: abcdefghijklmnopqrstuv18340
   b) пример неправильных выражений: abcdefghijklmnoasdfasdpqrstuv18340.
3. Дан текст со списками цен. Извлечь из него цены в USD, RUB, EU.
   a) пример правильных выражений: 25.98 USD.
   b) пример неправильных выражений: 44 ERR, 0.004 EU.
4. Дан текст, необходимо проверить есть ли в тексте цифры, за которыми не стоит знак «+».
   a) пример правильных выражений:(1 + 8) – 9 / 4
   b) пример неправильных выражений: 6 / 5 – 2 * 9

5. Написать регулярное выражение, определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1900 года до 9999 года.

a) пример правильных выражений: 29/02/2000, 30/04/2003, 01/01/2003.
b) пример неправильных выражений: 29/02/2001, 30-04-2003, 1/1/1899.

6. Написать регулярное выражение, определяющее является ли данная строчка допустимым (корректным) e-mail адресом согласно RFC под номером 2822.

a) пример правильных выражений: user@example.com, root@localhost
b) пример неправильных выражений: myhost@@@com.ru, @my.ru, Julia String.

7. Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть цифр, английская буква, и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

a) пример правильных выражений: F032_Password, TrySpy1.
b) пример неправильных выражений: smart_pass, A007.

8. Напишите метод filter, который принимает на вход массив (любого типа) и реализацию интерфейса Filter с методом apply(Object o), чтобы убрать из массива лишнее. Проверьте как он работает на строках или других объектах.

9. Постройте частотный словарь9 букв русского (или английского) алфавита. Здесь не имеет значения проблема выбора и анализа корпуса языка, достаточно будет взять текст небольшой длины).

[9]https://ru.wikipedia.org/wiki/%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%BD%D1%8B%D0%B9_%D1%81%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C

Практическая работа № 15. Вложенные и внутренние классы. Обработка событий в Джава программах с графическим интерфейсом пользователя

Цель данной практической работы - изучить использование анонимных и внутренних классов, научиться разрабатывать интерактивные программы на языке Джава с использованием графического интерфейса пользователя.

Теоретические сведения

В Java вложенные классы — это классы, которые определены внутри другого класса. Цель вложенного класса - четко сгруппировать вложенный класс с окружающим его классом, сигнализируя, что эти два класса должны использоваться вместе. Или, возможно, вложенный класс должен использоваться только изнутри его включающего (владеющего) класса. Разработчики на Джава часто ссылаются на вложенные классы в качестве внутренних классов, но внутренние классы (нестатические вложенные классы) только один из нескольких различных типов вложенных классов в Джава. В Джава вложенные классы считаются компонентами своего включающего класса. Таким образом, вложенный класс не может быть объявлен public, package (без модификатора доступа), protected и private (см. модификаторов доступа для получения дополнительной информации). Следовательно, вложенные классы в Джава также могут наследоваться подклассами. В Джава можно создать несколько различных типов вложенных классов.

Рассмотрим различные типы вложенных классов Java:

- Статические вложенные классы
- Нестатические вложенные классы
- Местные классы
- Анонимные классы

Статические вложенные классы

Статические вложенные классы объявляются в Java следующим образом:

```
public class Outer {
  public static class Nested {
  }}
```

Чтобы создать экземпляр Nested класса, вы должны сослаться на него, добавив к нему префикс имени Outer класса, например:

```
Outer.Nested instance = new Outer.Nested ();
```

Листинг 15.1 – Пример класса Question

```
public class Question {
  private Type type;
  public Type getType() { return type; }
  public void setType(Type type) { this.type = type; }

  public static enum Type {
  SINGLE_CHOICE, MULIT_CHOICE, TEXT
  }

}
```

В языке Джава статический вложенный класс — это, по сути, обычный класс, который только что был вложен в другой класс. Будучи статическим, статический вложенный класс может получить доступ только к переменным экземпляра включающего класса через ссылку на экземпляр включающего класса.

Нестатические вложенные классы

Нестатические вложенные классы в Java также называются внутренними классами. Внутренние классы связаны с экземпляром включающего класса. Таким образом, вы должны сначала создать экземпляр включающего класса, чтобы создать экземпляр внутреннего класса. Вот пример определения внутреннего класса:

```
public class Outer {
  public class Inner { //вложенный класс
  }
}
```

Вот как вы создаете экземпляр Inner класса:

```
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
```

Обратите внимание, как вы ставите new после ссылки на внешний класс, чтобы создать экземпляр внутреннего класса. нестатические вложенные классы (внутренние классы) имеют доступ к полям включающего класса, даже если они объявлены закрытыми. Вот пример:

Листинг 15.2 – Пример внутреннего (иннер) класса

```
public class Outer {
    private String text = "I am private!";
    public class Inner {
```

```
            public void printText() {
                System.out.println(text);
            }
        }
    }
```

Вложенные классы делятся на статические (в примере выше StaticNestedClass — это именно такой класс) и нестатические (non-static).

Собственно нестатические вложенные классы имеют и другое название - *внутренние классы (inner classes). Внутренний класс (inner)* является подмножеством вложенного класса (nested classes). *Внешний класс (outer class)* мы иногда будем называть еще обрамляющим классом.

Локальные классы (local inner classes)

Локальные классы (local classes) определяются в блоке кода в программе на языке Джаве. На практике чаще всего объявление происходит в методе некоторого другого класса. Хотя объявлять локальный класс можно внутри статических и нестатических блоков инициализации. Пример использования локального класса:

Листинг 15.3 – Пример локального класса

```
Public class Handler {
Public void handle(String requestPath){
class Path{
List<String> parts = new ArrayList<String>();
String path = "/";
Path(String path) {
if (path == null) return;
this.path = path;
for (String s : path.split("/"))
if(s.trim().length() > 0) this.parts.add(s);
}
int size(){return parts.size();}
String get(int i){
return i > this.parts.size() - 1 ? null :
this.parts.get(i);
}
boolean startsWith(String s){
return path.startsWith(s);
}
```

```
}
Path path = new Path(requestPath);
if(path.startsWith("/page")){
String pageId = path.get(1);
... }
if(path.startsWith("/post")){
   categoryId = path.get(1);
   String postId = path.get(2);
...
 }
...
}}
```

Данный код с некоторыми изменениями взят из реального проекта и используется для обработки get запросов к веб-серверу. Он вводит новую абстракцию, с которой удобно работать в пределах метода и которая не нужна за его пределами.

Как и обычный поля классы, локальные классы ассоциируются с экземпляром обрамляющего класса и имеют доступ к его полям и методам. Кроме этого, локальный класс может обращаться к локальным переменным и параметрам метода, если они объявлены с модификатором final.

У локальных классов есть множество ограничений:

- они видны только в пределах блока, в котором объявлены;
- они не могут быть объявлены как private, public, protected или static;
- они не могут иметь внутри себя статических объявлений (полей, методов, классов); исключением являются константы (static final);

Кстати, интерфейсы тоже нельзя объявлять локально по тем же причинам, по каким их нельзя объявлять внутренними.

Анонимные классы (anonymous inner classes)

Анонимные классы являются важным подспорьем в повседневной жизни программистов на языке Джава. Анонимный класс (anonymous class) — это локальный класс без имени. Классический пример анонимного класса:

```
new Thread(new Runnable() {
     public void run() {
          ...
     }
 }).start();
```

На основании анонимного класса создается поток и запускается с помощью метода start класса Thread. Синтаксис создания анонимного класса базируется на использовании оператора new с именем класса (интерфейса) и телом вновь создаваемого анонимного класса.

Основное ограничение при использовании анонимных классов — это невозможность описания конструктора, так как класс не имеет имени. Аргументы, указанные в скобках, автоматически используются для вызова конструктора базового класса с теми же параметрами. Вот пример:

Листинг 15.4 – Пример анонимного класса

```
Class Clazz {
Clazz(int param) { }
Public static void main(String[] args){
// правильное создание анонимного класса
New Clazz(1) { };
// неправильное создание анонимного класса
New Clazz() { };
}}
```

Так как анонимный класс является локальным классом, он имеет все те же ограничения, что и локальный класс.

Использование анонимных классов оправдано во многих случаях, в частности когда:

- тело класса является очень коротким;
- нужен только один экземпляр класса;
- класс используется в месте его создания или сразу после него;
- имя класса не важно и не облегчает понимание кода.

Остается только сказать, что нужно использовать с умом возможности, которые предоставляют нам вложенные классы, это сделает ваш код чище, красивее и понятнее.

Преимущества использования вложенного класса

Преимущества вложенных классов в языке Джава заключаются в том, что вы можете группировать классы вместе, которые принадлежат друг другу. Вы, конечно, можете группировать их, например поместив в один пакет, но размещение одного класса внутри другого делает такую группировку более сильной.

Вложенный класс, как правило, используется только через класс, в который он вложен. Иногда вложенный класс виден только классу, в который он вложен, используется только внутри и, следовательно, никогда не виден за

пределами этого класса. В других случаях вложенный класс виден за пределами класса, в который он вложен, но не может использоваться без него

Обработка событий в программах с графическим интерфейсом пользователя

Что означает термин обработка событий? В Джава это механизм, который контролирует состояние объекта на предмет наступления события и при его появлении запускает процедуру обработки события. В качестве объектов выступают различные компоненты графического интерфейса, например кнопки или текстовые поля. Это также может быть курсор мыши, нажатие кнопки мыши или нажатие клавиши клавиатуры. То есть, объекты gui это источники событий. Для каждого объекта создаётся слушатель события с помощью специального класса. Слушатель при наступлении события запускает процедуру обработки события.

Интерфейс ActionListener

В Джава есть интерфейс ActionListener. Он получает уведомление всякий раз, когда вы нажимаете кнопку или выбираете пункт меню. Он уведомлен о наступлении события ActionEvent. Интерфейс ActionListener находится в пакете java.awt.event . У него только один метод: actionPerformed (). При создании собственного обработчика вам нужно добавить в свой класс реализацию этого метода.

Метод actionPerformed ()

Синтаксис метода:

```
public abstract void actionPerformed (ActionEvent e);
```

Метод actionPerformed () вызывается автоматически всякий раз, когда вы изменяете состояние объекта, например щелкаете мышкой по кнопке. Как написать ActionListener в своем классе. Для этого вам необходимо выполнить 3 шага:

1. Реализуйте интерфейс ActionListener в классе:

```
public class ActionListenerExample implements
ActionListener
```

2. Зарегистрируйте компонент в Listener (component здесь это тот компонент для которого пишете):

```
component.addActionListener(instanceOfListenerclass);
```

3. Переопределите метод actionPerformed ()

```
public void actionPerformed(ActionEvent e){
//Здесь пишете свой код
}
```

Листинг 15.5 – Пример обработки нажатия на кнопку

```
import java.awt.*;
import java.awt.event.*;
//первый шаг
public class ActionListenerExample implements
ActionListener{
public static void main(String[] args) {
    Frame f=new Frame("ActionListener Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);
    //второй шаг
    b.addActionListener(this);
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
//третий шаг
public void actionPerformed(ActionEvent e){
            tf.setText("Добро пожаловать в мир
Джава.");
}
}
```

Результат выполнения программы на листинге 15.5 вы можете увидеть на рис. 15.1.

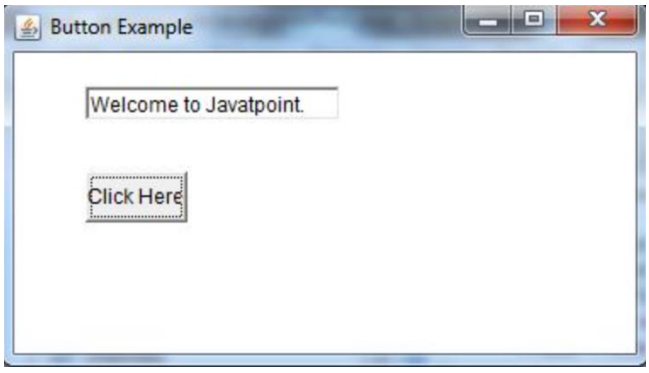

Рисунок 15.1. Пример работы программы на листинге 15.4

Использование анонимных классов

Рассмотрим пример реализации ActionListener через анонимный класс:

Мы также можем использовать анонимный класс для реализации

ActionListener, вместо способа, рассмотренного в листинге 15.1 Это сокращенный способ, поэтому вам теперь не нужно выполнять все три шага описанные выше. Добавляем слушателя к кнопке таким образом:

```
b.addActionListener ( новый  ActionListener () {
public void  actionPerformed (ActionEvent e) {
    tf.setText ( «Добро пожаловать в мир Java.» );
 }
});
```

Листинг 15.6 – Пример программы с анонимным классом

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample {
    public static void main(String[] args) {
        Frame f=new Frame("ActionListener Example");
        final TextField tf=new TextField();
        tf.setBounds(50,50, 250,20);
        Button b=new Button("Click Here");
        b.setBounds(50,100,60,30);
        b.setSize(100,50);

        b.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent e){
                tf.setText("Добро пожаловать в мир
Джава.");
            }
        });
        f.add(b);f.add(tf);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Результат работы программы на листинге 15.6 представлен на рис.15.2.

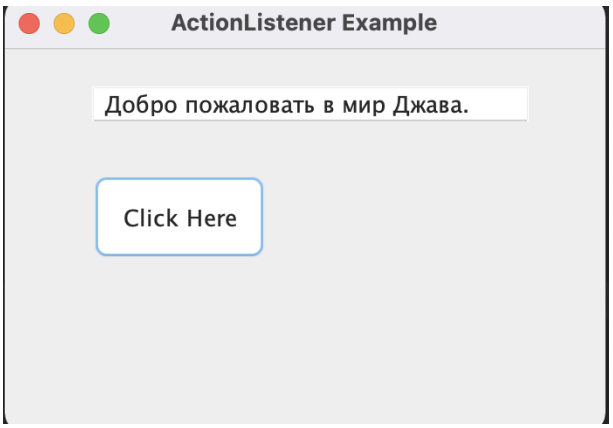

Рисунок 15.2 Пример работы программы на листинге 15.5

Компонент JTextArea

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

Листинг 15.7 – Пример с полем ввода текста

```
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(ActionEventae){
 String txt = JOptionPane.showInputDialog (
null,"Insert some text");
jta1.append(txt);
}
});
}
public static void main(String[]args){
new TextAreaExample().setVisible(true);
  }
}
```

На рис. 15.3 представлен пример выполнения программы, представленной на листинге 15.7.

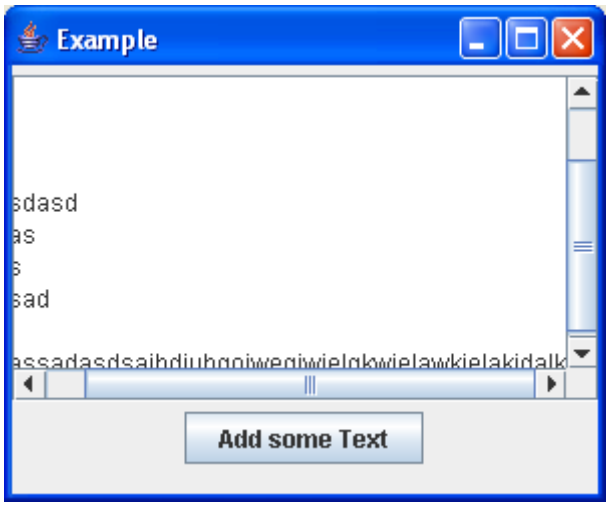

Рисунок 15.3. Пример работы программы на листинге 15.7

Рассмотрим еще один пример программы, для сложения двух чисел.

Листинг 15.8 – Пример программы сложения 2 чисел

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class LabExample extends JFrame{
JTextField jta1 = newJTextField(10);
JTextField jta2 = new JTextField(10);
JButton button = new JButton(" Add themup");
Font fnt = new Font("Times new roman",Font.BOLD,20);
//начало конструктора класса LabExample
LabExample(){
super("Example");
setLayout(new FlowLayout());
setSize(250,150);
// создаем надпись 1
add(new JLabel("1st Number"));
// добавляем текстовое поле jta1
add(jta1);
// создаем надпись 2

add(new JLabel("2nd Number"));
// добавляем текстовое поле jta2
add(jta2);
// добавляем кнопку
add(button);
```

```
    //добавляем слушателя к кнопке
    button.addActionListener(new ActionListener(){
    // добавляем реализацию actionPerformed
    public void actionPerformed(ActionEventae){
    try{
    //переменная для хранения ввода в текстовое поле 1
    double x1 = Double.parseDouble(jta1.getText().trim());
    //переменная для хранения ввода в текстовое поле 2
    double x2 = Double.parseDouble(jta2.getText().trim());
    //создаем всплывающее окно INFORMATION_MESSAGE
    JOptionPane.showMessageDialog(null, "Result =
    "+(x1+x2),"Alert",JOptionPane.INFORMATION_MESSAGE);
    }catch(Exception e){
    //создаем всплывающее окно ERROR_MESSAGE
    JOptionPane.showMessageDialog( null, "Error in Numbers
!","alert" , JOptionPane.ERROR_MESSAGE);
        }
     }
    });// конец button.addActionListener()
    } // конец конструктора
    setVisible(true);
    }
    public static void main(String[]args){
    new LabExample();
     } // конец main()
    } // конец класса LabExample
```

На рис. 15.4, 15.5, 15.6, 15.7 мы видим результат работы программы, представленной на листинге 15.8

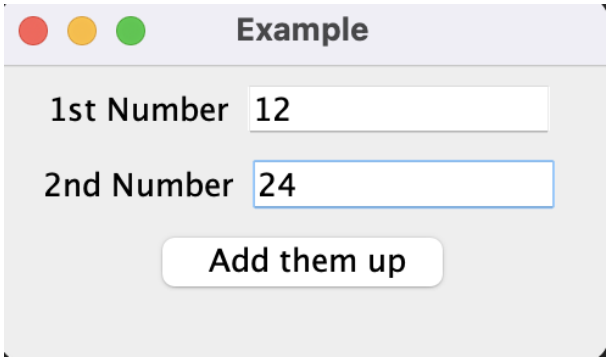

Рисунок 15.4. Пример работы программы на листинге 15.8

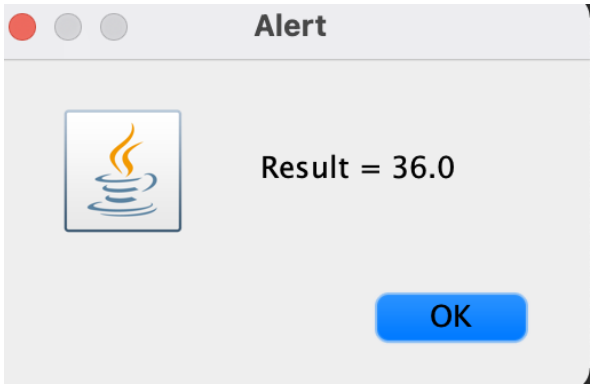

Рисунок 15.5. Пример работы программы на листинге 15.6

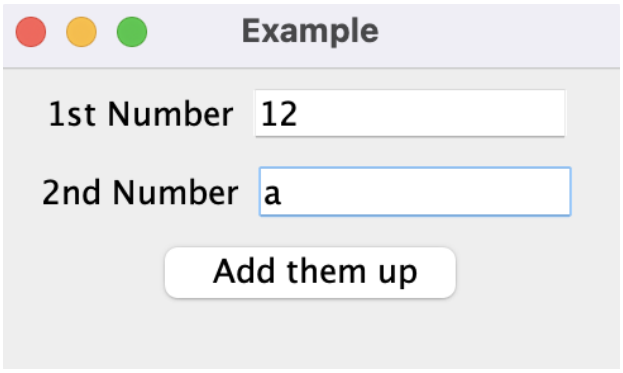

Рисунок 15.6. Пример работы программы на листинге 15.6

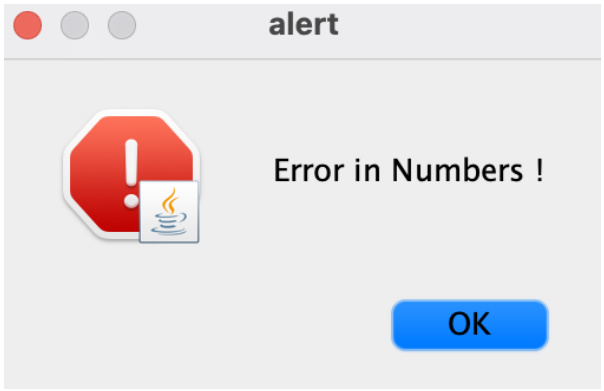

Рисунок 15.7. Пример работы программы на листинге 15.6

Замечание.

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

```
JTextArea txtArea = new JTextArea(20,20) ;
JScrollPane jScroll = new JScrollPane(txtArea);
// …
/* мы добавим полосу прокрутки панели (scrollPane) а не объект TextArea*/
add(Scroll);
```

Попробуйте выполнить сами!

Задания на практическую работу № 15

1. Напишите программу калькулятор, используя пример в листинге 15.6. Реализуйте помимо сложения вычитание, деление и умножение для двух чисел, которые вводятся с клавиатуры.

2. Разработайте программу выбора меню как на рис. 15.8 ниже. Вам понадобится JComboBox.

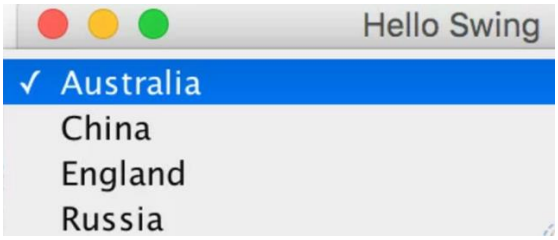

Рисунок 15.8. Окно выбора меню

При выборе пункта меню должна выводится информация о стране

3. Разработайте программу с меню, двумя кнопками и текстовым полем ввода. В этой программе у вас должны быть разные настройки в меню. Должно быть меню «Файл», которое включает в себя подменю «Сохранить», «Выйти» и «Правка», включая подменю «Копировать, вырезать, вставить» и меню «Справка». Вид окна программы должен иметь вид как на рис.15.9 ниже.

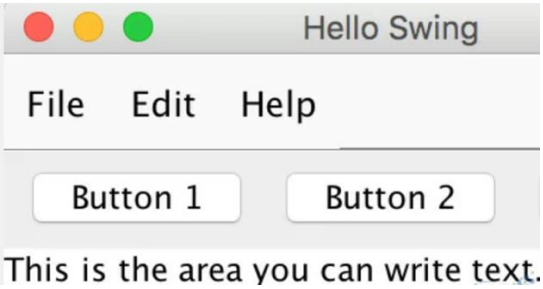

Рисунок 15.9. Окно выбора меню

Замечание. Для выполнения задания вам понадобятся следующие классы: GridLayout, JButton, JFrame, JMenu, JMenuBar, JMenuItem, JPanel, JTextArea;

4. Разработайте программу калькулятора вида как представлено на рис. 15.10 ниже

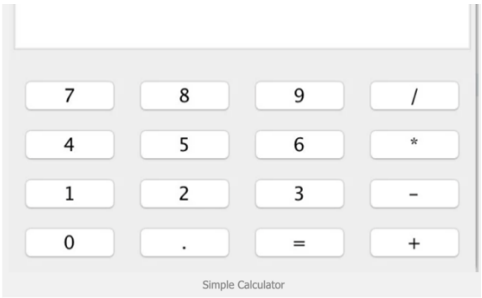

Рисунок 15.10. Окно программы калькулятор

Практическая работа № 16. Обработка событий мыши и клавиатуры программах на Джава с графическим интерфейсом пользователя

Цель: научиться обрабатывать различные события мыши и клавиатуры для разных компонентов.

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 мышивошел в область наблюдаемого объекта
- public void mouseExited(MouseEvent event) — курсор мышивышел из области наблюдаемого объекта
- public void mousePressed(MouseEvent event) — кнопка мышинажата в момент, когда курсор находится над наблюдаемым объектом
- public void mouseReleased(MouseEvent event) — кнопка мыши отпущена в момент, когда курсор находится наднаблюдаемым 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 используетсяметод 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 внутреннимполям 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 щелчкапо 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 теметоды, 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, когданаблюдаемый объект теряет фокус.

Слушатель колеса манипулятора мыши 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 символтипа 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.

Рассмотрим пример программы, где отрабатывается выбор меню с помощью нажатия пункта меню и программа закрывается по нажатию сочетания клавиш клавиатуры. Полный листинг с кодом программы приведен в приложении А.

На рис. 16.1 представлен свернутый код программы.

ShortCut.java

```
1   import javax.swing.AbstractAction;
2   import javax.swing.ImageIcon;
3   import javax.swing.JFrame;
4   import javax.swing.JMenu;
5   import javax.swing.JMenuBar;
6   import javax.swing.JMenuItem;
7   import javax.swing.KeyStroke;
8   import java.awt.EventQueue;
9   import java.awt.event.ActionEvent;
10  import java.awt.event.InputEvent;
11  import java.awt.event.KeyEvent;
12
13      public class ShortCut extends JFrame {
14
15          public ShortCut() {...}
19
20          private void initUI() {...}
29
30          private void createMenuBar() {...}
69
70          private class MenuItemAction extends AbstractAction {...}
86
87          public static void main(String[] args) {
88              /*   java.awt.EventQueue.invokeLater(new Runnable() {
89                       public void run() {
90                           new ShortCut().setVisible(true);
91                       }
92                   });
93                   */
94          EventQueue.invokeLater(() -> {
95
96                  var ex = new ShortCut();
97                  ex.setVisible(true);
98              });
99
100
101         }
102     }
103
104
```

Рисунок 16.1. Окно программы Меню выбора

Для обеспечения функциональности необходимо добавить выражения импорта для всех классов, которые мы будем использовать.

Соответственно это:

```
import javax.swing.AbstractAction;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
```

Класс ShortCut наследуется от базового класса JFrame. В нашем классе реализован конструктор ShortCut(){} класса, который состоит из одного метода объявленного ниже:

```
public ShortCut() {

    initUI();
}
```

Метод initUI() объявлен с модификатором private, это вспомогательный метод класса, так называемый утилитный. Код метода ниже:

```
private void initUI() {
    createMenuBar();
    setTitle("Выбор меню");
    setSize(360, 250);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
}
```

Задача метода создать базовое окно с заголовком “Выбор меню” и вызвать функцию создания меню createMenuBar(). Также наш метод initUI() делает следующее - установает размеры окна – метод setSize(360, 250), реакцию окна по умолчанию на закрытие, а метод setLocationRelativeTo(**null**) используется, чтобы центрировать окно на экране.

Метод createMenuBar() также является приватным, он используется только внутри класса. И выполняет очень важные функции по обработке событий мыши. Код метода представлен на листинге 16.1.

Обратите внимание, что при объявлении типов переменных для создания объектов графического интерфейса используется служебное слово var. На

первой лекции по языку Джава, когда мы обсуждали типы данных говорилось об этом типе. Такой тип означает local variable или локальная переменная. Ключевое слово var можно использовать только с локальными переменными, то есть переменными, которые объявлены, например внутри конструкторов или внутри блоков инициализации, или внутри методов.

Для создания меню используется класс JMenu. Мы создаем этот объект с помощью вызова конструктора JMenu("Файл"). Далее с помощью вызова конструктора класса MenuItemAction() создаются объекты newMenuItem, openMenuItem, saveMenuItem exitMenuItem.

Листинг 16.1 Метод createMenuBar()

```
private void createMenuBar() {

    var menuBar = new JMenuBar();

    var iconNew = new ImageIcon("src/resources/new.png");
    var iconOpen = new
ImageIcon("src/resources/open.png");
    var iconSave = new
ImageIcon("src/resources/save.png");
    var iconExit = new
ImageIcon("src/resources/exit.png");

    var fileMenu = new JMenu("Файл");
    fileMenu.setMnemonic(KeyEvent.VK_F);

    var newMenuItem = new JMenuItem(new
MenuItemAction("Создать файл", iconNew,
            KeyEvent.VK_N));

    var openMenuItem = new JMenuItem(new
MenuItemAction("Открыть файл", iconOpen,
            KeyEvent.VK_O));

    var saveMenuItem = new JMenuItem(new
MenuItemAction("Сохранить файл", iconSave,
            KeyEvent.VK_S));

    var exitMenuItem = new JMenuItem("Выход", iconExit);
```

```
        exitMenuItem.setMnemonic(KeyEvent.VK_E);
        exitMenuItem.setToolTipText("Выход из приложения");

exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEve
nt.VK_E,InputEvent.CTRL_DOWN_MASK));

        exitMenuItem.addActionListener((event) ->
System.exit(0));

        fileMenu.add(newMenuItem);
        fileMenu.add(openMenuItem);
        fileMenu.add(saveMenuItem);
        fileMenu.addSeparator();
        fileMenu.add(exitMenuItem);

        menuBar.add(fileMenu);

        setJMenuBar(menuBar);
}
```

Строчки кода ниже

```
var exitMenuItem = new JMenuItem("Выход", iconExit);
exitMenuItem.setMnemonic(KeyEvent.VK_E);
```

Переменная exitMenuItem представляет собой пункт меню выбрать который модно с помощью нажатия мыши и также по сочетанию клавиш для setMnemonic(KeyEvent.*VK_E*) означает нажатие <контрол+E>. При выборе меню файл и нажатии

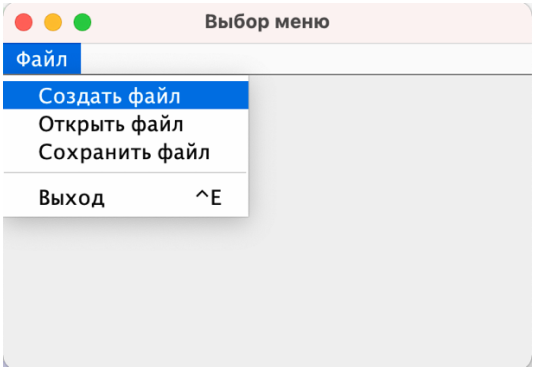

Рисунок 16.2 Окно программы Выбор меню.

Добавляем пункты меню с помощью метода add(Component comp):

```
fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(saveMenuItem);
```

```
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);
```

Затем добавляем

```
menuBar.add(fileMenu);
setJMenuBar(menuBar);
```

Также если включить в пункты меню пиктограммы

```
var menuBar = new JMenuBar();
var iconNew = new
ImageIcon("/Users/natala/Downloads/create_file.jpeg");
var iconOpen = new
ImageIcon("/Users/natala/Downloads/open_file.jpeg" );
var iconSave = new
ImageIcon("/Users/natala/Downloads/save_file.jpeg ");
var iconExit = new
ImageIcon("/Users/natala/Downloads/exit_prog.jpeg");
```

На рис.16.3 представлено окно меню с использованием пиктограмм

Рисунок 16.3 Окно программы Выбор меню с пиктограммами

И наконец, для того чтобы запустить программу в классе MenuItemAction, который наследуется от класса AbstractAction описан конструктор класса с тремя параметрами MenuItemAction(String text, ImageIcon icon, Integer mnemonic), Внутри конструктора происходит вызов конструктора родителя **super(**text и производится переопределение метода actionPerformed(ActionEvent e) для реализации реакции на событие.

Листинг 16.2 Класс MenuItemAction

```
private class MenuItemAction extends AbstractAction {

    public MenuItemAction(String text, ImageIcon icon,
                          Integer mnemonic) {
        super(text);

        putValue(SMALL_ICON, icon);
        putValue(MNEMONIC_KEY, mnemonic);
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        System.out.println(e.getActionCommand());
    }
}
```

В методе main() точке входа в программу следующий код:

```
java.awt.EventQueue.invokeLater(new Runnable() {
     public void run() {
         new ShortCut().setVisible(true);
     }
 });
```

Означает, что этом примере вы видите анонимный класс, который создается как **new** Runnable(). Этот анонимный класс переопределяет метод запуска интерфейса runnable. Затем этот анонимный класс создается и передается методу EventQueue.invokeLater(), который является статическим методом. Этот метод добавляет объект в очередь событий - eventQueue. В EvenQueue много событий, таких как события клавиатуры или события мыши или что-то еще какие-то события, ждущие обработки. Существует поток,

который продолжает опрос данных из этой очереди. После того, как этот поток достигнет анонимного класса, который был здесь создан, то он выполнит метод run(), который создаст объект класса ShortCut (пользовательский класс окна) и сделает его видимым.

Вся сложность этого участка кода заключается в том, что новая часть ShortCut.setVisible (true) не выполняется в основном потоке, а в случае диспетчерского потока. В Swing вы должны выполнить весь код, который изменяет пользовательский интерфейс в потоке диспетчеризации[10] событий.

Замечание. Поток диспетчеризации событий-это поток, который обрабатывает все события GUI и управляет вашим Swing GUI. Он всегда запускается где-то в коде Swing, если в вашей программе есть GUI. Причина, по которой это делается не явным образом, заключается в простоте - вам не нужно беспокоиться о запуске и управлении дополнительным потоком самостоятельно.

Важно помнить, что классы Swing не являются потокобезопасными. Это означает, что вы всегда должны вызывать методы Swing из одного и того же потока, иначе вы рискуете получить странное или неопределенное поведение. GUI может быть изменен только из одного потока, поэтому вы должны обновлять свой GUI с помощью метода invokeLater(), это связано с проблемами параллелизма. Когда пользователь нажимает кнопку на GUI, то Runnable , который уведомляет всех слушателей, прикрепленных к кнопке, и попадает в очередь. То же самое происходит когда пользователь меняет размер И когда вы используете invokeLater(), то экземпляр ваш Runnable попадает в очередь.

То же самое можно переписать, используя лямбда выражения

```
EventQueue.invokeLater(() -> {

        var ex = new ShortCut();
        ex.setVisible(true);
    });
```

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 каждогоконкретного компонента. В интерфейсе определен всего один метод:

[10] https://stackoverflow.com/questions/7217013/java-event-dispatching-thread-explanation

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) — окноразвернуто.

public void windowActivated(WindowEvent event) — окно сталоактивным.

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 обладаетодним 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 остальнымлишь в тех относительно редких случаях, когда возникнет такая необходимость.

Обработка нажатия на кнопку ok легко переписывается дляActionListener:

```
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 на практическую работу №16

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

а. Когда мышь входит в область CENTER программа показывает

диалоговое окно с сообщением “Добро пожаловать в ЦАО”

b. Когда мышь входит в область WEST программа показывает диалоговое окно с сообщением “Добро пожаловать в ЗАО”

c. Когда мышь входит в область SOUTH программа показывает диалоговое окно с сообщением “Добро пожаловать ЮАО”

d. Когда мышь входит в область NORTH программа показывает диалоговое окно с сообщением “Добро пожаловать в САО”

e. Когда мышь входит EAST программа показывает диалоговое окно с сообщением “Добро пожаловать в ВАО”

2. Реализуйте программу на Джава с использованием JTextArea и двумя следующего меню выбора:

a) Цвет: который имеет возможность выбора из три возможных: синий, красный и черный

b) Шрифт: три вида: “Times New Roman”, “MS Sans Serif”, “Courier New”.

Замечание. Вы должны написать программу, которая с помощью меню, может изменять шрифт и цвет текста, написанного в JTextArea

3. Реализуйте программу Проверка пароля на Джава с использованием Layout менеджеров компоновки. Окно программы должно иметь вид как на рис. 16.41.

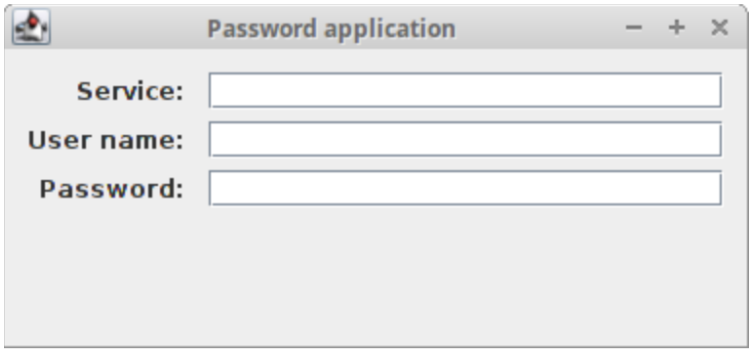

Рисунок 16.4 Окно программы Проверка пароля

Список литературы

1. The Java Tutorials, http://docs.oracle.com/javase/tutorial/index.html
2. История версий Java Edition [Электронный ресурс]. – Режим доступа: URL: https://minecraft.fandom.com/ru/wiki/%D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9_(Java_Edition) дата доступа: 15.08.2021.
3. Зорина Н.В. Объектно-ориентированный анализ и программирование [Электронный ресурс]: учебное пособие / Н. В. Зорина. – М.: РТУ МИРЭА, 2019 — 111 с.
4. Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн 11 издание. – М.: Издательский дом Вильямс, 2019 — 864 с., с ил.; ISBN 978-5-907114-79-1, 978-0-13-516630-7
5. Хорстманн Кей С. Java SE 9. Базовый курс. - Издательство Альфа-книга 2018 — 576 с., с ил.; ISBN: 978-0-13-469472-6
6. Блох Джошуа, Java: эффективное программирование 3-е издание. – М. : Издательский дом Вильямс, 2018 — 464 с., с ил.; ISBN 978-5-6041394-4-8, 978-0-13-468599-1
7. Брюс Эккель, Философия Java. Серия: Классика computer science, 4-е издание. Питер: 2019 — 1168 с., с ил.; ISBN: 978-5-4461-1107-7
8. Java Platform, Standard Edition Oracle JDK Migration Guide Release 13 F18399-01 September 2019 [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/java/javase/13/migrate/migration-guide.pdf, дата доступа: 12.05.2021.
9. JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/java/javase/index.html, дата доступа: 21.06.2021.
10. Спецификация языка Java 8 [Электронный ресурс]. – Режим доступа: URL: https://docs.oracle.com/javase/specs/jls/se8/html/index.html, дата доступа: 11.12.2021
11. Зорина Н.В. Объектно-ориентированное программирование [Электронный ресурс]: конспект лекций/ Н. В. Зорина. – М.: РТУ МИРЭА, 2019 — 119 с.
12. Зорина Н.В. Объектно-ориентированный анализ и программирование [Электронный ресурс]: учебное пособие / Н. В. Зорина. — М.: РТУ МИРЭА, 2019. — Электрон. опт. диск (ISO) https://library.mirea.ru/share/3240

13. Зорина Н.В. Объектно-ориентированное программирование на Java [Электронный ресурс]: метод. рекомендации / Н. В. Зорина [и др.]. — М.: РТУ МИРЭА, 2018. — Электрон. опт. диск (ISO) https://library.mirea.ru/share/2995

14. Объектно-ориентированное программирование на Java [Электронный ресурс]: практикум / Н. В. Зорина. — М.: РТУ МИРЭА, 2019. — Электрон. опт. https://library.mirea.ru/share/3185

15. Кишори, Ш. Java 9. Полный обзор нововведений. Для быстрого ознакомления и миграции / Ш. Кишори; перевод с английского А.А. Слинкин. — Москва: ДМК Пресс, 2018. — 544 с. — ISBN 978-5-97060-575-2. — Текст: электронный // Электронно-библиотечная система «Лань» : [сайт]. — URL: https://e.lanbook.com/book/108130 (дата обращения: 19.11.2019). — Режим доступа: для авториз. пользователей.

16. Васильев Алексей Николаевич Java. Объектно-ориентированное программирование: Базовый курс по объектно-ориентированному программированию: для магистров и бакалавров / А. Н. Васильев. — СПб.: Питер, 2014. — 397 с.: ил. — (Учебное пособие). — Библиогр.: с. 377

17. https://javarush.ru/groups/posts/458-key-khorstmann-i-ego-1500-slov-o-tom-kak-statjh-luchshim-java-programmistom--

ПРИЛОЖЕНИЕ А. Листинг программы 16.1

```
import javax.swing.AbstractAction;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;

    public class ShortCut extends JFrame {

        public ShortCut() {

            initUI();
        }

        private void initUI() {

            createMenuBar();

            setTitle("Выбор меню");
            setSize(360, 250);
            setLocationRelativeTo(null);
            setDefaultCloseOperation(EXIT_ON_CLOSE);
        }

        private void createMenuBar() {

            var menuBar = new JMenuBar();

            var iconNew = new
ImageIcon("src/resources/new.png");
            var iconOpen = new
ImageIcon("src/resources/open.png");
            var iconSave = new
ImageIcon("src/resources/save.png");
            var iconExit = new
ImageIcon("src/resources/exit.png");

            var fileMenu = new JMenu("Файл");
```

```
            fileMenu.setMnemonic(KeyEvent.VK_F);

            var newMenuItem = new JMenuItem(new
MenuItemAction("Создать файл", iconNew,
                    KeyEvent.VK_N));

            var openMenuItem = new JMenuItem(new
MenuItemAction("Открыть файл", iconOpen,
                    KeyEvent.VK_O));

            var saveMenuItem = new JMenuItem(new
MenuItemAction("Сохранить файл", iconSave,
                    KeyEvent.VK_S));

            var exitMenuItem = new JMenuItem("Выход",
iconExit);
            exitMenuItem.setMnemonic(KeyEvent.VK_E);
            exitMenuItem.setToolTipText("Выход из
приложения");

exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEve
nt.VK_E,
                    InputEvent.CTRL_DOWN_MASK));

            exitMenuItem.addActionListener((event) ->
System.exit(0));

            fileMenu.add(newMenuItem);
            fileMenu.add(openMenuItem);
            fileMenu.add(saveMenuItem);
            fileMenu.addSeparator();
            fileMenu.add(exitMenuItem);

            menuBar.add(fileMenu);

            setJMenuBar(menuBar);
        }

        private class MenuItemAction extends
AbstractAction {

            public MenuItemAction(String text, ImageIcon
icon,
                                  Integer mnemonic) {
                super(text);
```

```
                putValue(SMALL_ICON, icon);
                putValue(MNEMONIC_KEY, mnemonic);
            }

            @Override
            public void actionPerformed(ActionEvent e) {

                System.out.println(e.getActionCommand());
            }
        }

        public static void main(String[] args) {
         /*   java.awt.EventQueue.invokeLater(new
Runnable() {
                public void run() {
                    new ShortCut().setVisible(true);
                }
            });
            */
        EventQueue.invokeLater(() -> {

                var ex = new ShortCut();
                ex.setVisible(true);
            });


        }
    }
```