|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 1

по дисциплине «Объектно-ориентированное программирование»

«ЦИКЛЫ, УСЛОВИЯ, ПЕРЕМЕННЫЕ И МАССИВЫ В JAVA»

ВАРИАНТ 4

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «12» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

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

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

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

**Создание программы на Java.**

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

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

Пример:

|  |
| --- |
| package example;  public class Example {  public static void main(String[] args) {  }  } |

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

**Переменные.**

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

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

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

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

Пример:

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

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

**Массивы.**

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

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

|  |
| --- |
| int[] b = new int[10]; |

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

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

**Условия.**

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

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

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

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

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

**Циклы.**

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

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

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

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

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

|  |
| --- |
| while(logic) {  //Тело цикла  } |

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

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

|  |
| --- |
| do {  //Тело цикла  }while(logic); |

**Потоки ввода/вывода и строки в Java, класс String**

Для ввода данных используется класс Scanner из библиотеки пакетов

Этот класс надо импортировать в той программе, где он будет

использоваться. Это делается до начала открытого класса в коде программы.

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

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

System.err

|  |
| --- |
| importjava.util.Scanner; // импортируем класс  importjava.util.Scanner; // импортируем класс  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in); // создаём объект  класса Scanner  int i = 2;  System.out.print("Введитецелоечисло: "); if(sc.hasNextInt()) { // возвращаетистиннуеслиспотока  ввода можно считать целое число  i = sc.nextInt(); // считывает целое число с потока ввода и сохраняем в переменную  System.out.println(i\*2);  } else {  System.out.println("Вы ввели не целое число");  }  }  } |

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

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

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

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

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

строке под индексом n (индекс это порядковый номер символа, но нумероваться символы начинают снуля).

|  |
| --- |
| public class Main {  public static void main(String[] args) {  String s1 = "firefox";  System.out.println(s1.toUpperCase()); // выведет  «FIREFOX»  String s2 = s1.replace('o', 'a'); System.out.println(s2); // выведет «firefax» System.out.println(s2.charAt(1)); // выведет «i» int i;  i = s1.length(); System.out.println(i); // выведет 7 i = s1.indexOf('f'); System.out.println(i); // выведет 0 i = s1.indexOf('r'); System.out.println(i); // выведет 2 i = s1.lastIndexOf('f'); System.out.println(i); // выведет 4 i = s1.indexOf('t'); System.out.println(i); // выведет -1 i = s1.indexOf('r',3); System.out.println(i); // выведет -1  }  } |

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

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

**Методы в языке Java.**

Методы позволяют выполнять блок кода, из любого другого места, где это доступно. Методы определяются внутри классов. Методы могут быть статическими(можно выполнять без создания экземпляра класса), не статическими (не могут выполняться без создания экземпляра класса). Методы могут быть открытыми(public), закрытыми(private). Закрытые методы могут вызываться только внутри того класса, в котором они определены. Открытые методы можно вызывать для объекта внутри других классов.

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

Пример:

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

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

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

|  |
| --- |
| int s = sum(10,15); |

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* сгенерировать массив целых чисел случайным образом, вывести его на экран, отсортировать его, и снова вывести на экран.

*Решение:*

*Листинг 1 — Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  import java.util.Random;  import java.util.Arrays;  public class Main {  public static void main(String[] args) {  Random rnd = new Random();  int n = rnd.nextInt(15 - 4) - 1;  int[] nums = new int [n];  for (int i = 0;i<n-1;i++){  nums[i] = rnd.nextInt(20);  }  System.out.println(Arrays.toString(nums));  //сортировка  int k;  for(int j = 0; j < n-1; j++){  if (nums[j] > nums[j+1]) {  k = nums[j+1];  nums[j+1] = nums[j];  nums[j] = k;  j = -1;  }  }  System.out.println(Arrays.toString(nums));  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы, были получены начальные навыки работы со средой разработки IntelliJ IDEA и языком программирования Java.

Также были изучены базовые понятия массивов и случайных чисел.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 2

по дисциплине «Объектно-ориентированное программирование»

«ООП в java. понятие класса»

ВАРИАНТ 1

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

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

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

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

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

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

**Создание классов в Java.**

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

|  |
| --- |
| <модификатор досупа> class <имя класса> {  <тело класса>  } |

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

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

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

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

В отличие от метода, конструктор никогда ничего не возвращает.

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

|  |
| --- |
| public class Rectangle {  //Свойства, поля класса  private float width;  private float height;  //Конструктор класса  public Rectangle(float w, float h) {  width=w;  height=h;  }    //Метод, возвращающий ширину прямоугольника  public float getWidth() {  return width;  }  //Метод, возвращающий высоту прямоугольника  public float getHeight() {  return height;  }  //Метод, устанавливающий ширину прямоугольника  public void setWidth(float w) {  width=w;  }  //Метод, устанавливающий высоту прямоугольника  public void setHeight(float h) {  height=h;  }  } |

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

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

**Создание экземпляра класса.**

Для того чтобы создать экземпляр класса необходимо объявить переменную, тип которой соответствует имени класса или суперкласса. После чего нужно присвоить этой переменной значение, вызвав конструктор создаваемого класса с помощью оператора new. Например, можно создать экземпляр класса Rectangle следующим образом:

|  |
| --- |
| Rectangle rect = new Rectangle(20, 10); |

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

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

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  public class Main {  public static void main(String[] args) {  Human Vitya = new Human();  Human Zhenya = new Human();  System.out.println("Витя:");  Vitya.head.form\_Head = "Квадратная";  System.out.println(Vitya.head.form\_Head);  Vitya.head.think("кушать");  System.out.println(Vitya.hand.fingers);  Vitya.hand.skills("бить, держать");  Vitya.leg.size = 44;  System.out.println(Vitya.leg.size);  Vitya.leg.cheto();  //Витя кончился |

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java) – продолжение*

|  |
| --- |
| System.out.println("Женя:");  System.out.println(Zhenya.head.form\_Head);  Zhenya.head.think("поспать");  System.out.println(Zhenya.hand.fingers);  Zhenya.hand.skills("писать, театр теней");  System.out.println(Zhenya.leg.size);  Zhenya.leg.cheto();  }  } |

*Листинг 2 – Класс Human (файл Human.java)*

|  |
| --- |
| package com.company;  public class Human {  Hand hand = new Hand();  Head head = new Head();  Leg leg = new Leg();  } |

*Листинг 3 – Head, объект этого класса создаётся в классе Human (файл Head.java)*

|  |
| --- |
| package com.company;  public class Head {  public String form\_Head = "Овальная";  public void think(String wish) {  System.out.println("Хочет " + wish);  }  } |

*Листинг 4 – Leg, объект этого класса создаётся в классе Human (файл Leg.java)*

|  |
| --- |
| package com.company;  public class Leg {  public int size = 41;  public void cheto(){  System.out.println("Ходить, бегать, прыгать");  }  } |

*Листинг 5 – Hand, объект этого класса создаётся в классе Human (файл Hand.java)*

|  |
| --- |
| package com.company;  public class Hand {  public int fingers = 5;  public void skills(String skill){  System.out.println("Может " + skill);  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были получены представления об объектно-ориентированном программировании в Java. Также получены начальные знания о понятии класса и его создании.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 3

по дисциплине «Объектно-ориентированное программирование»

«НАСЛЕДОВАНИЕ В java»

ВАРИАНТ 1

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

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

|  |
| --- |
| public class Person {    private String name;  private String surname;    public String getName() { return name; }  public String getSurname() { return surname; }    public Person(String name, String surname){    this.name=name;  this.surname=surname;  }    public void displayInfo(){    System.out.println("Имя: " + name + " Фамилия: " + surname);  }  } |

И, возможно, впоследствии мы решили расширить имеющуюся систему и классов, добавив в нее класс, описывающий сотрудника предприятия - класс Employee. Так как этот класс реализует тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником) от класса Person, который, в свою очередь, называется базовым классом или родителем:

|  |
| --- |
| class Employee extends Person{    } |

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

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

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

|  |
| --- |
| class Employee extends Person{    private String company;    public Employee(String name, String surname, String company) {    super(name, surname);  this.company=company;  }    public void displayInfo(){    super.displayInfo();  System.out.println("Компания: " + company);  }  } |

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

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

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

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

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

|  |
| --- |
| public void displayInfo(){  System.out.println("Имя: " + super.getName() + " Фамилия: "  + super.getSurname() + " Компания: " + company);  } |

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

|  |
| --- |
| public static void main(String[] args) {    Employee empl = new Employee("Tom", "Simpson", "Oracle");  empl.displayInfo();  String firstName = empl.getName();  System.out.println(firstName);  } |

Хотя наследование очень интересный и эффективный механизм, но в некоторых ситуациях его применение может быть нежелательным. И в этом случае можно запретить наследование с помощью ключевого слова final. Например:

|  |
| --- |
| public final class Person {  } |

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

|  |
| --- |
| class Employee extends Person{  } |

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

|  |
| --- |
| public class Person {    //........................    public final void displayInfo(){    System.out.println("Имя: " + name + " Фамилия: " + surname);  }  } |

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

**Абстрактные классы**

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

При определении абстрактных классов используется ключевое слово abstract:

|  |
| --- |
| public abstract class Human{  private int height;  private double weight;  public int getHeight() { return height; }  public double getWeight() { return weight; }  } |

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

|  |
| --- |
| public abstract void displayInfo(); |

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

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

|  |
| --- |
| public abstract class Person {  private String name;  private String surname;  public String getName() { return name; }  public String getSurname() { return surname; }  public Person(String name, String surname){  this.name=name;  this.surname=surname;  }  public abstract void displayInfo();  }    class Employee extends Person{    private String bank;    public Employee(String name, String surname, String company) {    super(name, surname);  this.bank=company;  }  public void displayInfo(){    System.out.println("Имя: " + super.getName() + " Фамилия: "  + super.getSurname() + " Работает в банке: " + bank);  }  }  class Client extends Person  {  private String bank;    public Client(String name, String surname, String company) {    super(name, surname);  this.bank=company;  }  public void displayInfo(){    System.out.println("Имя: " + super.getName() + " Фамилия: "  + super.getSurname() + " Клиент банка: " + bank);  }  } |

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* создать абстрактный класс, описывающий собак(Dog). С помощью наследования реализовать различные породы собак. Протестировать работу классов.

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  public class Main {  public static void main(String[] args) {  Chihua chihua = new Chihua("Средней длины", "миниматюрный", "Чихуа-Хуа");  Chow chow = new Chow("Длинношёрстные", "Большие", "Чау-Чау");  Staff staff = new Staff("Короткошёрстные", "Средние", "Английский стаффордширский бультерьер");  chihua.displayInfo();  chow.displayInfo();  staff.displayInfo();  }  } |

*Листинг 2 – Абстрактный класс Dog (файл Dog.java)*

|  |
| --- |
| package com.company;  public abstract class Dog {  private String wool;  private String size;  public String getWool() { return wool; }  public String getSize() { return size; }  public Dog(String wool, String size){  this.wool=wool;  this.size=size;  }  public void displayInfo(){  System.out.println("Шерсть: " + wool + " Размер: " + size + "\n");  }  } |

*Листинг 3 – Chow, класс-наследник (файл Chow.java)*

|  |
| --- |
| package com.company;  class Chow extends Dog {  private String breed;  public Chow(String wool, String size, String breed) {  super(wool, size);  this.breed=breed;  }  public void displayInfo(){  System.out.println("Порода: " + breed);  super.displayInfo();  }  } |

*Листинг 4 –Staff, класс-наследник (файл Staff.java)*

|  |
| --- |
| package com.company;  class Staff extends Dog {  private String breed;  public Staff(String wool, String size, String breed) {  super(wool, size);  this.breed=breed;  }  public void displayInfo(){  System.out.println("Порода: " + breed);  super.displayInfo();  }  } |

*Листинг 5 – Chihua, класс-наследник (файл Chihua.java)*

|  |
| --- |
| package com.company;  class Chihua extends Dog {  private String breed;  public Chihua(String wool, String size, String breed) {  super(wool, size);  this.breed=breed;  }  public void displayInfo(){  System.out.println("Порода: " + breed);  super.displayInfo();  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были изучены понятия наследования, родительских и абстрактных классов в Java.

Также были закреплены навыки работы с классами и их взаимодействием.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 4

по дисциплине «Объектно-ориентированное программирование»

«ИНТЕРФЕЙСЫ в java»

ВАРИАНТ 2

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

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

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

Чтобы определить интерфейс, используется ключевое слово interface.

Определим следующий интерфейс:

|  |
| --- |
| public interface Printable{    void print();  } |

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

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

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

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

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

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

Чтобы класс применил интерфейс, надо использовать ключевое слово implements:

|  |
| --- |
| class Book implements Printable{    String name;  String author;  int year;    Book(String name, String author, int year){  this.name = name;  this.author = author;  this.year = year;  }    public void print() {    System.out.printf("Книга '%s' (автор %s) была издана в %d году \n", name, author, year);  }  } |

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

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

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

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

|  |
| --- |
| Printable pr = new Printable();  pr.print(); |

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

|  |
| --- |
| public class Journal implements Printable {    private String name;    String getName(){  return name;  }    Journal(String name){    this.name = name;  }  public void print() {  System.out.printf("Журнал '%s'\n", name);  }  } |

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

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

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

|  |
| --- |
| public static void main(String[] args) {    Printable printable = createPrintable("Компьютерра",false);  printable.print();    read(new Book("Отцы и дети", "И. Тургенев", 1862));  read(new Journal("Хакер"));  }    static void read(Printable p){    p.print();  }    static Printable createPrintable(String name, boolean option){    if(option)  return new Book(name, "неизвестен", 2015);  else  return new Journal(name); |

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

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

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* реализовать интерфейс Priceable, имеющий метод getPrice(), возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  public class Main {  public static void main(String[] args) {  Book book = new Book(269);  book.getPrice();  Pen pen = new Pen(14);  pen.getPrice();  Shirt shirt = new Shirt(899);  shirt.getPrice();  }  } |

*Листинг 2 – ИнтерфейсPriceable (файл Priceable.java)*

|  |
| --- |
| package com.company;  public interface Priceable {  int getPrice();  } |

*Листинг 3 – Класс Book (файл Book.java)*

|  |
| --- |
| package com.company;  public class Book {  int price;  Book(int price){  this.price=price;  }  public void getPrice(){  System.out.println("Книга - " + price + " рублей");  }  } |

*Листинг 4 – Класс Shirt (файл Shirt.java)*

|  |
| --- |
| package com.company;  public class Shirt {  int price;  Shirt(int price){  this.price=price;  }  public void getPrice(){  System.out.println("Рубашка - " + price + " рублей");  }  } |

*Листинг 5 – Класс Pen (файл Pen .java)*

|  |
| --- |
| package com.company;  public class Pen {  int price;  Pen(int price){  this.price=price;  }  public void getPrice(){  System.out.println("Ручка - " + price + " рублей");  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были получены начальные знания об интерфейсах в Java, их реализации и использовании в программах.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 5

по дисциплине «Объектно-ориентированное программирование»

«СОЗДАНИЕ ПРОГРАММ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ ПОЛЬЗОВАТЕЛЯ НА java»

ВАРИАНТ 2

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

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

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

Пример содания объекта класса JTextField:

JTextField jta = new JTextField (10);

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

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

|  |
| --- |
| class LabExample extends JFrame  {  JTextField jta = new JTextField(10);  Font fnt = new Font("Times new roman",Font.BOLD,20); LabExample()  {  super("Example"); setLayout(new FlowLayout()); setSize(250,100);  add(jta); jta.setForeground(Color.PINK); jta.setFont(fnt); setVisible(true);  }  public static void main(String[]args)  {  new LabExample();  }  } |



**Важное замечание**

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

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

|  |
| --- |
| import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  class LabExample extends JFrame  {  JTextField jta1 = new JTextField(10); JTextField jta2 = new JTextField(10);  JButton button = new JButton(" Add them up");  Font fnt = new Font("Times new roman",Font.BOLD,20); LabExample()  {  super("Example"); setLayout(new FlowLayout()); setSize(250,150);  add(new JLabel("1st Number")); add(jta1);  add(new JLabel("2nd Number")); add(jta2);  add(button);  button.addActionListener(new ActionListener()  {  public void actionPerformed(ActionEvent ae)  {  try  {  double x1 = Double.parseDouble(jta1.getText().trim());  double x2 = Double.parseDouble(jta2.getText().trim());  JOptionPane.showMessageDialog(null, "Result = "+(x1+x2),"Alert",JOptionPane.INFORMATION\_MESSAGE);  }  catch(Exception e)  {  JOptionPane.showMessageDialog(null, "Error in Numbers !","alert" , JOptionPane.ERROR\_MESSAGE);  }  }  });  setVisible(true);  }  public static void main(String[]args)  {  new LabExample();  }  } |

**JTextArea**

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

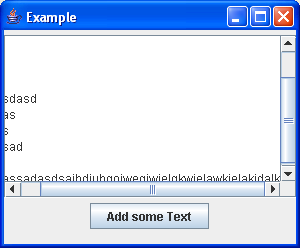
|  |
| --- |
| import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  class TextAreaExample extends JFrame  {  JTextArea jta1 = new JTextArea(10,25);  JButton button = new JButton("Add some Text");  public TextAreaExample()  {  super("Example");  setSize(300,300);  setLayout(new FlowLayout()); add(jta1);  add(button);  button.addActionListener(new ActionListener()  {  public void actionPerformed(ActionEvent ae)  {  String txt = JOptionPane.showInputDialog(null,"Insert some text");  jta1.append(txt);  }  });  }  public static void main(String[]args)  {  new TextAreaExample().setVisible(true);  }  } |

**Замечание**

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

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

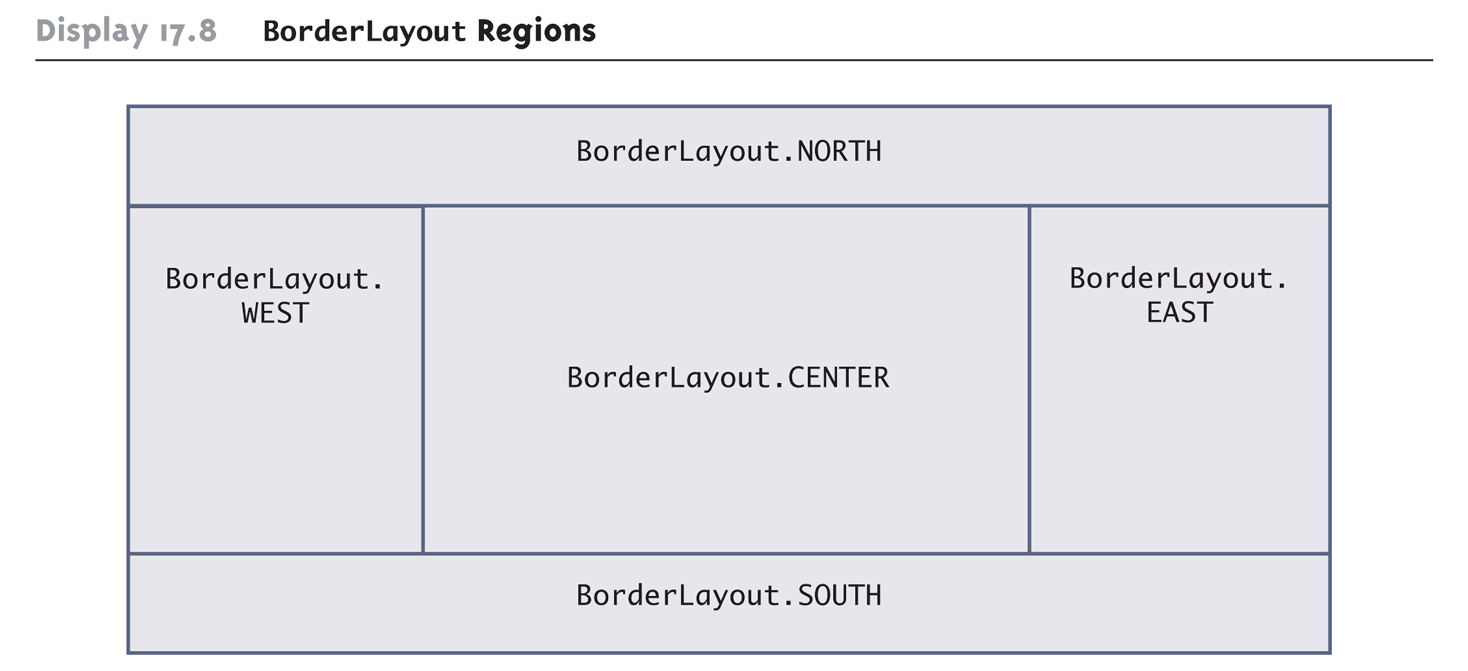
Попробуйте выполнить сами!



**Layout Менеджеры:**

**BorderLayout:**

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



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

add( comp , BorderLayout.EAST);

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

**GridLayout менеджер**

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

1 2 3 4

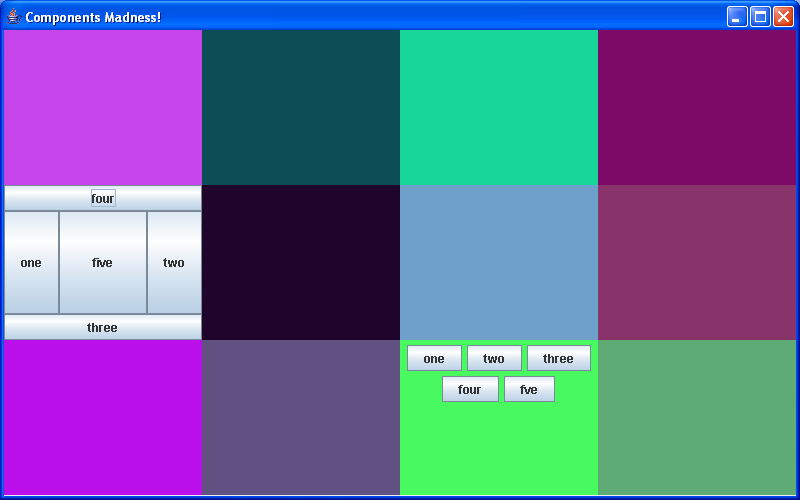
5 6 7 8

9 10 11 12

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

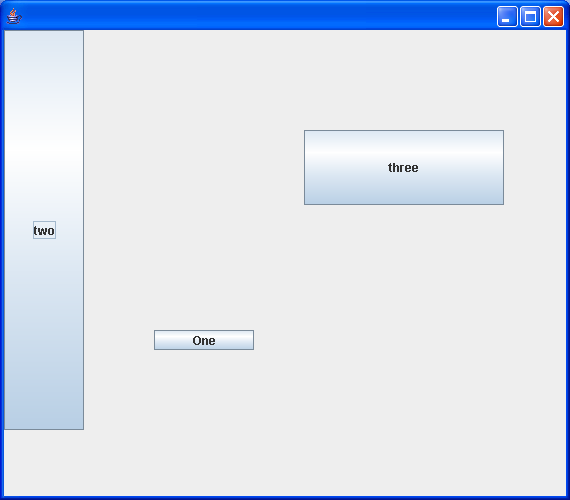
Следующий пример иллюстрирует смесь компоновки различных компонентов

|  |
| --- |
| import javax.swing.\*; import java.awt.\*; importjava.awt.event.\*;  class BorderExample extends JFrame  {  JPanel[] pnl = new JPanel[12];  public BorderExample()  {  setLayout(new GridLayout(3,4)); for(int i = 0 ; i < pnl.length ; i++)  {  int r = (int) (Math.random() \* 255); int b = (int) (Math.random() \* 255); int g = (int) (Math.random() \* 255); pnl[i] = new JPanel(); pnl[i].setBackground(new Color(r,g,b)); add(pnl[i]);  }  pnl[4].setLayout(new BorderLayout()); pnl[4].add(new JButton("one"),BorderLayout.WEST); pnl[4].add(new JButton("two"),BorderLayout.EAST);  pnl[4].add(new JButton("three"),BorderLayout.SOUTH); pnl[4].add(new JButton("four"),BorderLayout.NORTH); pnl[4].add(new JButton("five"),BorderLayout.CENTER);  pnl[10].setLayout(new FlowLayout()); pnl[10].add(new JButton("one"));  pnl[10].add(new JButton("two"));  pnl[10].add(new JButton("three"));  pnl[10].add(new JButton("four"));  pnl[10].add(new JButton("fve"));  setSize(800,500);  }  public static void main(String[]args)  {  new BorderExample().setVisible(true);  }  } |

Вот такой будет иметь вид, представленный выше код

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

**Null Layout Manager**

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

|  |
| --- |
| import javax.swing.\*; import java.awt.\*; importjava.awt.event.\*;  class NullLayout extends JFrame  {  JButton but1 = new JButton("One");; JButton but2 = new JButton("two");; JButton but3 = new JButton("three");;  public NullLayout()  {  setLayout(null);  but1.setBounds(150,300,100,20); // added at 150,300 width = 100, height=20  but2.setSize(80,400); // added at 0,0 width = 80, height=400  but3.setLocation(300,100); but3.setSize(200,75);  // those two steps can be combined in one setBounds method call  add(but1); add(but2); add(but3); setSize(500,500);  }  public static void main(String[]args)  {  new NullLayout().setVisible(true);  }  } |

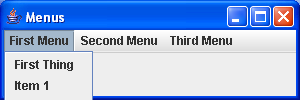


**Меню**

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

• JMenuBar: который представляет собой компонент, который содержит меню.

• JMenu: который представляет меню элементов для выбора.

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

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

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* cоздать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки.

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| import java.awt.\*;  import javax.swing.\*;  import java.util.Scanner;  public class Main {  public static void main(String[] args){  String n;  Scanner in = new Scanner(System.in);  n = in.next();  new Window(n);  }  static class Window extends JFrame{  Window(String n){  setSize(400, 600);  setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);  setLayout(new FlowLayout());  ImageIcon icon = new ImageIcon(n); |

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java) – продолжение*

|  |
| --- |
| setIconImage(icon.getImage());  getContentPane().add(new  JLabel(icon)).setPreferredSize(new Dimension(400, 600));  setLocationRelativeTo(null);  setVisible(true);  }  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были получены начальные навыки работы с графическим интерфейсом пользователя, формами, свойствами и объектами формы.

Также были получены навыки работы с командной строке при программировании на Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 6

по дисциплине «Объектно-ориентированное программирование»

«ОБРАБОТКА СОБЫТИЙ В JAVA ПРОГРАММАХ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ ПОЛЬЗОВАТЕЛЯ»

ВАРИАНТ 2

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «17» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

**Механизм обработки событий библиотеки Swing.**

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

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

**Интерфейс MouseListener и обработка событий от мыши.**

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

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

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

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

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

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

Мы сделали слушателя вложенным классом класса SimpleWindow, чтобы он мог легко получить доступ к его внутренним полям loginField и passwordField. Кроме того, хотя реально мы обрабатываем только одно из пяти возможных событий мыши, описывать пришлось все пять методов (четыре имеют пустую реализацию). Дело в том, что в противном случае класс пришлось бы объявить абстрактным (ведь он унаследовал от интерфейса пустые заголовки методов) и мы не смогли бы создать объект этого класса. А мы должны создать объект слушателя и прикрепить его к кнопке. Для этого в код конструктора SimpleWindow() необходимо добавить команду:

ok.addMouseListener(new MouseL());

Это можно сделать сразу после команды:

JButton ok = new JButton("OK");

Создание слушателей с помощью анонимных классов

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

ok.addMouseListener(new MouseL());

заменить на:

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

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

**Класс MouseAdapter**

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

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

В результате предыдущее описание слушателя будет выглядеть более компактно:

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

**Общая структура слушателей**

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

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

Чтобы привязать слушателя к объекту (который поддерживает соответствующий тип слушателей) используется метод add\*\*\*Listener(\*\*\*Listener listener).

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

**Слушатель фокуса FocusListener**

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

Интерфейс FocusListener имеет два метода:

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

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

**Слушатель колесика мышки MouseWheelListener**

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

public void mouseWheelMoved(MouseWheelEvent event).

**Слушатель клавиатуры KeyListener**

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

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

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

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

Аргумент event этих методов способен дать весьма ценные сведения. В частности, команда event.getKeyChar() возвращает символ типа char, связанный с нажатой клавишей. Если с нажатой клавишей не связан никакой символ, возвращается константа CHAR\_UNDEFINED. Команда event.getKeyCode() возвратит код нажатой клавиши в виде целого числа типа int. Его можно сравнить с одной из многочисленных констант, определенных в классе KeyEvent: VK\_F1, VK\_SHIFT, VK\_D, VK\_MINUS и т.д. Методы isAltDown(), isControlDown(), isShiftDown() позволяют узнать, не была ли одновременно нажата одна из клавиш-модификаторов Alt, Ctrl или Shift.

**Слушатель изменения состояния ChangeListener**

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

public void stateChanged(ChangeEvent event).

**Слушатель событий окна WindowListener**

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

public void windowOpened(WindowEvent event) — окно открылось.

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

public void windowClosed(WindowEvent event) — окно закрылось.

public void windowIconified(WindowEvent event) — окно свернуто.

public void windowDeiconified(WindowEvent event) — окно развернуто.

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

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

**Слушатель событий компонента СomponentListener**

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

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

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

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

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

**Слушатель выбора элемента ItemListener**

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

public void itemStateChanged(ItemEvent event).

**Универсальный слушатель ActionListener**

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

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

public void actionPerformed(ActionEvent event).

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

Обработка нажатия на кнопку ok легко переписывается для ActionListener:

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

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* реализация приложения Java, который имеет макет границы и надписи для каждой области в макете. Теперь определим события мыши, чтобы описать действия:

a. Когда мышь входит CENTER программа показывает диалоговое окно

(Добро пожаловать в )

b. Когда мышь входит WEST программа показывает диалоговое окно

(Добро пожаловать в Джидда)

c. Когда мышь входит SOUTH программа показывает диалоговое окно

(Добро пожаловать Абха)

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

e. Когда мышь входит EAST программа показывает диалоговое окно

(Добро пожаловать в Дахране)

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  import java.awt.BorderLayout;  import java.awt.Dimension;  import javax.swing.\*;  public class Main extends JFrame {  public Main() {  super("Main");  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  JPanel mainPanel = new JPanel();  mainPanel.setLayout(new BorderLayout());  JPanel buttonsPanel = new JPanel();  buttonsPanel.setLayout(new BorderLayout());  JButton butN = new JButton("NORTH");  JButton butS = new JButton("SOUTH");  JButton butE = new JButton("EAST");  JButton butW = new JButton("WEST");  JButton butC = new JButton("CENTER");  buttonsPanel.add(butN, BorderLayout.NORTH);  buttonsPanel.add(butS, BorderLayout.SOUTH);  buttonsPanel.add(butE, BorderLayout.EAST);  buttonsPanel.add(butW, BorderLayout.WEST);  buttonsPanel.add(butC, BorderLayout.CENTER);  butN.addMouseListener(new CustomListener());  butS.addMouseListener(new CustomListener());  butE.addMouseListener(new CustomListener()); |

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java) – продолжение*

|  |
| --- |
| butW.addMouseListener(new CustomListener());  butC.addMouseListener(new CustomListener());  getContentPane().add(buttonsPanel);  }  public static void main(String[] args) {  javax.swing.SwingUtilities.invokeLater(new Runnable() {  public void run() {  JFrame.setDefaultLookAndFeelDecorated(true);  JFrame frame = new Main();  frame.setPreferredSize(new Dimension(500, 140));  frame.pack();  frame.setLocationRelativeTo(null);  frame.setVisible(true);  }  });  }  } |

*Листинг 2 – Класс CustomListener (файл CustomListener.java)*

|  |
| --- |
| package com.company;  import javax.swing.\*;  import java.awt.event.MouseEvent;  import java.awt.event.MouseListener;  public class CustomListener implements MouseListener {  public void mouseClicked(MouseEvent e) {}  public void mouseExited(MouseEvent e) {}  public void mousePressed(MouseEvent e) {}  public void mouseReleased(MouseEvent e) {}  public void mouseEntered(MouseEvent e) {  JButton button = (JButton) e.getSource();  String text = button.getText();  JOptionPaneMouse b = new JOptionPaneMouse(text);  }  } |

*Листинг 3 – JoptionPaneMouse,отвечает за действие мыши(файл JOptionPaneMouse.java)*

|  |
| --- |
| package com.company;  import javax.swing.\*;  import java.awt.\*;  public class JOptionPaneMouse extends JFrame {  public JOptionPaneMouse(String text){  String itog = "";  switch (text) {  case "CENTER":  itog = "Добро пожаловать в";  break;  case "WEST":  itog = "Добро пожаловать в Джидда";  break;  case "SOUTH":  itog = "Добро пожаловать Абха";  break;  case "NORTH":  itog = "Добро пожаловать в";  break;  case "EAST":  itog = "Добро пожаловать в Дахране";  break;  }  JLabel lab = new JLabel(itog);  JPanel contents = new JPanel();  contents.setLayout(new BorderLayout());  lab.setHorizontalAlignment(JLabel.CENTER);  contents.add(lab, BorderLayout.CENTER);  setContentPane(contents);  setSize(220, 90);  setVisible(true);  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были получены навыки обработки различных событий компонентов графического интерфейса.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 7

по дисциплине «Объектно-ориентированное программирование»

«КОЛЛЕКЦИИ, ОЧЕРЕДИ, СПИСКИ в java»

ВАРИАНТ 2

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

**Коллекции.**

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

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

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

* **Collection**: базовый интерфейс для всех коллекций и других интерфейсов коллекций
* **Queue**: наследует интерфейс Collection и представляет функционал для структур данных в виде очереди
* **Deque**: наследует интерфейс Queue и представляет функционал для двунаправленных очередей
* **List**: наследует интерфейс Collection и представляет функциональность простых списков
* **Set**: также расширяет интерфейс Collection и используется для хранения множеств уникальных объектов
* **SortedSet**: расширяет интерфейс Set для создания сортированных коллекций
* **NavigableSet**: расширяет интерфейс SortedSet для создания коллекций, в которых можно осуществлять поиск по соответствию
* **Map**: предназначен для созданий структур данных в виде словаря, где каждый элемент имеет определенный ключ и значение. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection

Эти интерфейсы частично реализуются абстрактными классами:

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

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

* **ArrayList**: простой список объектов
* **LinkedList**: представляет связанный список
* **ArrayDeque**: класс двунаправленной очереди, в которой мы можем произвести вставку и удаление как в начале коллекции, так и в ее конце
* **HashSet**: набор объектов или хеш-множество, где каждый элемент имеет ключ - уникальный хеш-код
* **TreeSet**: набор отсортированных объектов в виде дерева
* **LinkedHashSet**: связанное хеш-множество
* **PriorityQueue**: очередь приоритетов
* **HashMap**: структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение
* **TreeMap**

**Класс ArrayList.**

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

ArrayList имеет следующие конструкторы:

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

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

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

* **void add(int index, E obj)**: добавляет в список по индексу index объект obj
* **boolean addAll(int index, Collection<? extends E> col)**: добавляет в список по индексу index все элементы коллекции col. Если в результате добавления список был изменен, то возвращается true, иначе возвращается false
* **E get(int index)**: возвращает объект из списка по индексу index
* **int indexOf(Object obj)**: возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1
* **int lastIndexOf(Object obj)**: возвращает индекс последнего вхождения объекта obj в список. Если объект не найден, то возвращается -1
* **E remove(int index)**: удаляет объект из списка по индексу index, возвращая при этом удаленный объект
* **E set(int index, E obj)**: присваивает значение объекта obj элементу, который находится по индексу index
* **void sort(Comparator<? super E> comp)**: сортирует список с помощью компаратора comp
* **List<E> subList(int start, int end)**: получает набор элементов, которые находятся в списке между индексами start и end

Используем класс ArrayList и некоторые его методы в программе:

|  |
| --- |
| import java.util.ArrayList;    public class CollectionApp {    public static void main(String[] args) {    ArrayList<String> states = new ArrayList<String>();  // добавим в список ряд элементов  states.add("Германия");  states.add("Франция");  states.add("Великобритания");  states.add("Испания");  states.add(1, "Италия"); // добавляем элемент по индексу 1    System.out.println(states.get(1));// получаем 2-й объект  states.set(1, "Дания"); // установка нового значения для 2-го объекта    System.out.printf("В списке %d элементов \n", states.size());  for(String state : states){    System.out.println(state);  }    if(states.contains("Германия")){    System.out.println("Список содержит государство Германия");  }    // удалим несколько объектов  states.remove("Германия");  states.remove(0);    Object[] countries = states.toArray();  for(Object country : countries){    System.out.println(country);  }  }  } |

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

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

Метод size() позволяет узнать количество объектов в коллекции.

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

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

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

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

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

**Класс LinkedList.**

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

Класс LinkedList имеет следующие конструкторы:

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

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

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

Рассмотрим применение связанного списка:

|  |
| --- |
| package collectionapp;    import java.util.LinkedList;    public class CollectionApp {    public static void main(String[] args) {    LinkedList<String> states = new LinkedList<String>();    // добавим в список ряд элементов  states.add("Германия");  states.add("Франция");  states.addLast("Великобритания"); // добавляем на последнее место  states.addFirst("Испания"); // добавляем на первое место  states.add(1, "Италия"); // добавляем элемент по индексу 1    System.out.printf("В списке %d элементов \n", states.size());  System.out.println(states.get(1));  states.set(1, "Дания");  for(String state : states){    System.out.println(state);  }  // проверка на наличие элемента в списке  if(states.contains("Германия")){    System.out.println("Список содержит государство Германия");  }    states.remove("Германия");  states.removeFirst(); // удаление первого элемента  states.removeLast(); // удаление последнего элемента    LinkedList<Person> people = new LinkedList<Person>();  people.add(new Person("Mike"));  people.addFirst(new Person("Tom"));  people.addLast(new Person("Nick"));  people.remove(1); // удаление второго элемента    for(Person p : people){    System.out.println(p.getName());  }  Person first = people.getFirst();  System.out.println(first.getName()); // вывод первого элемента  }  }  class Person{    private String name;  public Person(String value){    name=value;  }  String getName(){return name;}  } |

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

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* протестировать работу коллекции LinkedList.

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  import java.util.LinkedList;  public class Car {  String model;  public Car(String model) {  this.model = model;  }  public static void main(String[] args) {  LinkedList<Car> cars = new LinkedList<>();  Car kia = new Car("Kia Optima");  Car chevrolet = new Car("Chevrolet Camaro SS");  Car ford = new Car("Ford Mustang");  Car buggati = new Car("Buggati Veyron");  Car toyota = new Car("Toyota Mark 2");  Car nissan = new Car("Nissan GT-R");  cars.add(kia);  cars.add(chevrolet);  cars.add(ford);  System.out.println(cars);  cars.addFirst(buggati);  cars.addLast(toyota);  System.out.println(cars);  cars.remove(buggati);  cars.removeFirst();  System.out.println(cars);  cars.add(1, nissan);  cars.remove(2);  System.out.println(cars);  }  @Override  public String toString() {  return "Car: " + "model - " + model;  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были получены навыки работы с LinkedList.

Также были получены знания об очередях, списках и коллекциях, используемых в программах Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

ЛАБОРАТОРНАЯ РАБОТА № 8

по дисциплине «Объектно-ориентированное программирование»

«РАБОТА С ФАЙЛАМИ»

ВАРИАНТ 2

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «14» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:**

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

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

*Запись файлов. Класс FileWriter*

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

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

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

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

Запишем в файл какой-нибудь текст:

|  |
| --- |
| import java.io.\*;  public class FilesApp {  public static void main(String[] args) {  try(FileWriter writer = new FileWriter("C:\\SomeDir\\notes3.txt", false))  {  // запись всей строки  Stringtext = "Мама мыла раму, раму мыла мама";  writer.write(text);  // запись по символам  writer.append('\n');  writer.append('E');  writer.flush();  }  catch(IOException ex){  System.out.println(ex.getMessage());  }  }  } |

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

*Чтение файлов. Класс FileReader*

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

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

|  |
| --- |
| FileReader(String fileName)  FileReader(File file)  FileReader(FileDescriptor fd) |

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

|  |
| --- |
| import java.io.\*;  public class FilesApp {  public static void main(String[] args) {  try(FileReader reader = new FileReader("C:\\SomeDir\\notes3.txt"))  {  // читаемпосимвольно  int c;  while((c=reader.read())!=-1){  System.out.print((char)c);  }  }  catch(IOException ex){  System.out.println(ex.getMessage());  }  }  } |

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* реализовать вывод информации из файла на экран

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл Main.java)*

|  |
| --- |
| package com.company;  import java.io.\*;  public class Main {  public static void main(String[] args) {  BufferedReader br;  try {  String text;  br = new BufferedReader(new InputStreamReader(new FileInputStream  ("text.txt"), "windows-1251"));  while ((text = br.readLine()) != null){System.out.println(text);}  }  catch (IOException e) {System.out.println("Error" + e);}  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения лабораторной работы были получены навыки работы с файлами и чтением данных из фалов с использованием Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 1

по дисциплине «Объектно-ориентированное программирование»

«КЛАССЫ, КАК НОВЫЕ ТИПЫ ДАННЫХ. ПОЛЯ ДАННЫХ И МЕТОДЫ»

ВАРИАНТ 2

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**Цель работы**

Цель данной практической работы – освоить на практике работу с классами на Java.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**1. Понятие класса**

В Java, класс является определением объектов одного и того же вида. Другими словами, класс — это тип данных, создаваемый программистом для решения задач. Он представляет из себя шаблон, или прототип, который определяет и описывает статические свойства и динамическое поведение, общие для всех объектов одного и того же вида.

Экземпляр класса - реализация конкретного объекта типа класс. Другими словами, экземпляр экземпляра класса. Все экземпляры класса имеют аналогичные свойства, как задано в определении класса. Например, вы можете определить класс с именем "Студент " и создать три экземпляра класса "Студент": " Петр", " Павел" и " Полина ". Термин "Объект " обычно относится к экземпляру класса. Но он часто используется свободно, которые могут относиться к классу или экземпляру.

Графически можно представить класс в виде UML[[1]](#footnote-1) диаграммы как прямоугольник в виде как трех секций, в котором присутствует секция наименования класса, секция инкапсуляции данных и методов (функций или операций) класса. Пример общего представления диаграммы класса представлен на рисунке 1.1.

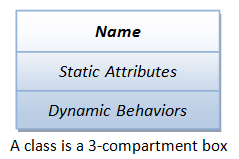


Рисунок 1.1 - Диаграмма класса. Общее представление.

Рассмотрим подробнее диаграмму класса. Имя (или сущность): определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа c данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“контейнере” или “боксе”), представленном на рисунке в виде прямоугольника.

На рисунке 1.2 показано несколько примеров классов. У каждого из них есть имя, переменные класса и методы.

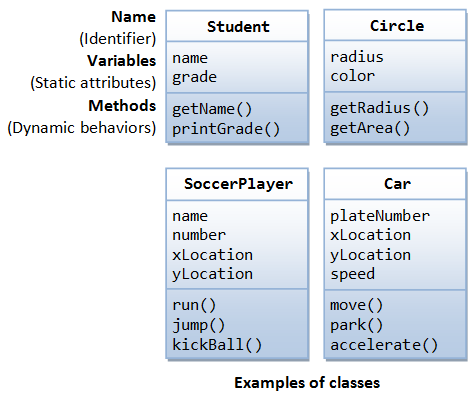


Рисунок 1.2 - Примеры классов

На рисунке 1.3 показаны два экземпляра класса типа Student "paul" и "peter" в виде UML диаграммы экземпляра класса.

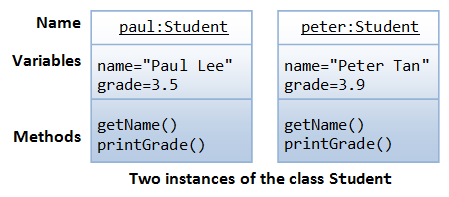


Рисунок 1.3 - Экземпляры класса Student.

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три отсека, один содержит название, два вторых переменные (поля данных класса) и методы, соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект класса) также представляется в виде прямоугольника, разделенного на три отсека, в первом отсеке, надпись с именем экземпляра, показанной в instanceName: Classname и выделенная подчеркиванием (название\_экземпляра : имя\_класса).

Кратко подведем итоги по определению класса:

1) Класс, тип данных, определяемый программистом, абстрактный тип данных, повторно-используемый программный объект, который имитирует реальные сущности предметной области. Класс можно представить графически в виде контейнера на UML диаграмме, который состоит из трех условных частей и содержит имя, переменные и методы.

2) Класс инкапсулирует статическое состояние объекта, его атрибуты или свойства данных в виде переменных класса и поведение объекта в виде методов, которые могут реализовывать определенные алгоритмы.

3) Значения переменных или поля данные составляют его состояние. Методы создает свои модели поведения.

Экземпляр класса — это представление (или реализация) конкретного представителя класса.

**2. Определение класса.**

В Java, мы используем слово class как ключевое или служебное слово, например, чтобы задать определение класса.

Для примера:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **public** **class** **Circle** { // class name  **double** radius; // variables  String color;    **double** **getRadius**() {...} // methods  **double** **getArea**() {...}  }  **public** **class** **SoccerPlayer** { // class name  **int** number; // variables  String name;  **int** x, y;    **void** **run**() {...} // methods  **void** **kickBall**() {...}  } |

Синтаксис определения класса в Java:

|  |  |
| --- | --- |
| 1  2  3  4 | [AccessControlModifier] **class** **ClassName** {  // class body contains definition of variables and methods  ...  } |

Давайте разъясним, что такое контроль доступа или спецификатор доступа, например, public и private, позже.

**Конвенция кода для класса (Class Naming Convention).**

Конвенцией кода называют соглашение между программистами о правилах написания кода. Соглашение содержит правила именования переменных и не только. Например, в соответствии с конвенцией кода на Java имя класса должно быть всегда существительным или словосочетанием из нескольких слов. Все слова должны с прописной буквы (так, называемая верблюжья нотация или camel notation). Совет: для имени класса всегда используйте существительное в единственном числе. Выберите значимое и самодостаточное имя для названия класса. Для примера, SoccerPlayer, HttpProxyServer, FileInputStream, PrintStream and SocketFactory будут подходящими именами в определенной предметной области, для которой вы пишете программу.

**3. Создание экземпляров класса**

Чтобы создать экземпляр класса, вы должны выполнить следующие действия:

объявить идентификатор экземпляра (имя экземпляра) конкретного класса.

Сконструировать экземпляр класса (то есть выделить память для экземпляра и инициализировать его) с помощью оператора "new".

Например, предположим, что у нас есть класс с именем Circle , тогдамы можем создавать экземпляры класса Circle, следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // Declare 3 instances of the class Circle, c1, c2, and c3  Circle c1, c2, c3;  // Allocate and construct the instances via new operator  c1 = **new** Circle();  c2 = **new** Circle(**2.0**);  c3 = **new** Circle(**3.0**, "red");  // You can declare and construct in the same statement  Circle c4 = **new** Circle(); |

**4. Операция получения доступа к компонентам класса.**

Доступ к компонентам класса осуществляется с помощью операции получения доступа, а именно операции точка “.”

Переменные и методы, входящие в состав класса, формально называется переменные-поля данных класса и методы класса и являются компонентами класса. Для ссылки на переменную-поле данных класса или метод, вы должны:

сначала создать экземпляр класса, который вам нужен;

затем, использовать оператор точка “.” чтобы сослаться на элемент класса (переменную-поле данных или метод класса).

Предположим, что у нас есть класс с именем Circle, с двумя переменными (радиус и цвет) и двумя методами (getRadius () и GetArea ()). Мы создали три экземпляра класса Circle, а именно, C1, C2 и C3 . Чтобы вызвать метод GetArea (), вы должны сначала определить к какой именно сущности вы обращаетесь, об этом собственно говорит говорит c2, а затем использовать оператор точка, в виде c2.getArea (), для вызова метода GetArea () экземпляра с2.

Например,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // Declare and construct instances c1 and c2 of the class Circle  Circle c1 = **new** Circle ();  Circle c2 = **new** Circle ();  // Invoke member methods for the instance c1 via dot operator  System.out.println(c1.getArea());  System.out.println(c1.getRadius());  // Reference member variables for instance c2 via dot operator  c2.radius = **5.0**;  c2.color = "blue" |

Вызов метода getArea () без указания экземпляра не имеет смысла , так как радиус неизвестно какого объекта (может быть много окружностей, у каждой из которых п свой собственный радиус) .

В общем, полагают, есть класс, называемый AClass с переменной-полем данных под названием aVariable и способом доступа к полю методом aMethod(). Экземпляр называется anInstance и строится с использованием AClass.

Вы можете использовать для доступа к открытым полям и методам операцию точка “.”, например - anInstance.aVariable и anInstance.aMethod().

**5. Переменные - поля данных класса**

Переменная-поле данных имеет имя (идентификатор) и тип, а также может иметь значение определенного типа, например базового или типа определенного программистом ранее. Переменная-поле данных может также быть экземпляром определенного класса (которые будут обсуждаться позже).

Конвенция об именах переменных гласит: имя переменной должно быть существительным или словосочетанием из нескольких слов. Первое слово в нижнем регистре, а остальные слова пишутся с прописной буквы (двугорбая нотация или camel notation), например, roomNumber, Xmax , Ymin и xTopLeft.

Обратите внимание, что имя переменной начинается с буквы в нижнем регистре, в то время как имя класса всегда начинается с заглавной буквы.

Формальный синтаксис для определения переменной в Java:

[модификатор\_доступа] тип имя\_перем [= иниц\_знач];

[модификатор\_доступа] тип имя\_пер-1 [=иниц\_знач-1], тип имя\_пер-2 [=иниц\_знач-2] ...;

Например:

|  |  |
| --- | --- |
| 1  2 | **private** **double** radius;  **public** **int** length = **1**, width = **1**; |

**6. Методы класса**

Метод (способ как описано в предыдущем разделе):

принимает параметры из вызова (как в функции);

выполняет операции, описанные в теле метода, и;

возвращает часть результата (или void) в точку вызова.

Формальный синтаксис объявления метода в Java:

|  |  |
| --- | --- |
| 1  2  3  4 | [AccessControlModifier] returnType methodName ([argumentList]) {  // method body or implementation  ......  } |

Например:

|  |  |
| --- | --- |
| 1  2  3 | **public** **double** **getArea**() {  **return** radius\*radius\*Math.PI;  } |

Конвенция кода о правилах записи имен методов гласит следующее: имя метода должно быть глаголом или начинаться глаголом в виде фразы из нескольких слов, первое слово записывается в нижнем регистре, а остальные слова начинаются с прописной буквы (двугорбая запись). Например, getRadius (), getParameterValues ().

Обратите внимание, что имя переменной существительное (обозначающий статический атрибут), в то время как имя метода - глагол (обозначает действие). Они имеют те же наименования. Тем не менее, вы можете легко отличить их от контекста. Методы могут принимать аргументы в скобках (возможно, нулевой аргумент, в пустых скобках), ноне поля данных. При записи, методы обозначаются парой круглых скобок, например, println(), getArea ().

**7. Теперь соберем все вместе: Пример ООП**

На рисунке ниже представлена диаграмма класса и его трех экземпляров.

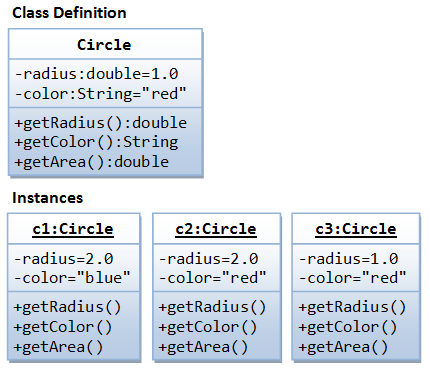


Рисунок 1.4 – Диаграмма класса и его экземпляров.

Класс называется Circle и должен быть определен, как показано на диаграмме классов выше.

Он содержит две переменные: radius (типа double) и color (типа String); и три метода: getRadius() , getColor(), и getArea () .

Три экземпляра Circle называются C1, C2, C3, и должны быть построены с учетом их соответствующих элементов данных и методов, как показано на схемах UML для экземпляров класса.

Исходные коды для изображенного на UML диаграмме класса Circle.java:

**Circle.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | // Define the Circle class  **public** **class** **Circle** { // Save as "Circle.java"  // Private variables  **private** **double** radius;  **private** String color;    // Constructors (overloaded)  **public** **Circle**() { // 1st Constructor  radius = **1.0**;  color = "red";  }  **public** **Circle**(**double** r) { // 2nd Constructor  radius = r;  color = "red";  }  **public** **Circle**(**double** r, String c) { // 3rd Constructor  radius = r;  color = c;  }    // Public methods  **public** **double** **getRadius**() {  **return** radius;  }  **public** String **getColor**() {  **return** color;  }  **public** **double** **getArea**() {  **return** radius\*radius\*Math.PI;  }  } |

Компиляция "Circle.java" в "Circle.class". Обратите внимание, что в классе Circle нет метода main(). Следовательно, это не будет программой на Java, и вы не можете запустить класс Circle сам по себе. Класс Circle нужен, чтобы быть отдельным строительным блоком и использоваться в других программах.

Дополним нашу программу еще одним классом, который будет демонстрировать работу с нашим классом. Мы напишем TestCircle, в котором будем использовать Circle класс. Класс TestCircle содержит метод main(), теперь мы можем откомпилировать и запустить программу.

**TestCircle.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **public** **class** **TestCircle** { // Save as "TestCircle.java"  **public** **static** **void** **main**(String[] args) { // Execution entry point  // Construct an instance of the Circle class called c1  Circle c1 = **new** Circle(**2.0**, "blue"); // Use 3rd constructor  System.out.println("Radius is " + c1.getRadius() // use dot operator to invoke member methods  + " Color is " + c1.getColor()  + " Area is " + c1.getArea());    // Construct another instance of the Circle class called c2  Circle c2 = **new** Circle(**2.0**); // Use 2nd constructor  System.out.println("Radius is " + c2.getRadius()  + " Color is " + c2.getColor()  + " Area is " + c2.getArea());    // Construct yet another instance of the Circle class called c3  Circle c3 = **new** Circle(); // Use 1st constructor  System.out.println("Radius is " + c3.getRadius()  + " Color is " + c3.getColor()  + " Area is " + c3.getArea());  }  } |

Запустим TestCircle и увидим результат:

Radius is 2.0 Color is blue Area is 12.566370614359172

Radius is 2.0 Color is red Area is 12.566370614359172

Radius is 1.0 Color is red Area is 3.141592653589793

**8. Конструкторы**

Конструктор – это специальный метод класса, который имеет то же имя, что используется в качестве имени класса. В приведенном выше класса Circle, мы определим три перегруженных версии конструктора Circle(...). Конструктор используется для создания и инициализации всех переменных-полей данных класса. Чтобы создать новый экземпляр класса, вы должны использовать специальный оператор "new" с последующим вызовом одного из конструкторов.

Например,

|  |  |
| --- | --- |
| 1  2  3 | Circle c1 = **new** Circle();  Circle c2 = **new** Circle(**2.0**);  Circle c3 = **new** Circle(**3.0**, "red"); |

Конструктор отличается от обычного метода следующим:

название метода-конструктора совпадает с именем класса, а имя класса по конвенции, начинается с заглавной буквы;

конструктор не имеет возвращаемого значения типа (или неявно не возвращает), таким образом, нет объявления типа возвращаемого значения при объявлении;

конструктор может быть вызван только через оператор «new», он может быть использован только один раз, чтобы инициализировать построенный экземпляр.

вы не можете впоследствии вызвать конструктор в теле программы подобно обычным методам (функциям);

конструкторы не наследуется (будет объяснено позже).

Конструктор без параметров называется конструктором по умолчанию, который инициализирует переменные-поля данных через их значения по умолчанию. Например, конструктор Circle() в рассмотренном выше примере.

**9.  Перегрузка методов**

Перегрузка методов означает, что несколько методов могут иметь то же самое имя метод, но сами методы могут иметь различные реализации (версии). Тем не менее, различные реализации должны быть различимы по списку их аргументов (либо количество аргументов, или типа аргументов, или их порядок).

Пример: метод average() имеет три версии с различными списками аргументов. При вызове может использоваться соответствующий выбору вариант, в соответствии с аргументами.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **public** **class** **TestMethodOverloading** {  **public** **static** **int** **average**(**int** n1, **int** n2) { // A  **return** (n1+n2)/**2**;  }    **public** **static** **double** **average**(**double** n1, **double** n2) { // B  **return** (n1+n2)/**2**;  }    **public** **static** **int** **average**(**int** n1, **int** n2, **int** n3) { // C  **return** (n1+n2+n3)/**3**;  }  **public** **static** **void** **main**(String[] args) {  System.out.println(average(**1**, **2**)); // Use A  System.out.println(average(**1.0**, **2.0**)); // Use B  System.out.println(average(**1**, **2**, **3**)); // Use C  System.out.println(average(**1.0**, **2**)); // Use B - int 2 implicitly casted to double 2.0  // average(1, 2, 3, 4); // Compilation Error - No matching method  }  } |

Рассмотрим перегрузка конструктора класса Circle.

Приведенный выше класс Circle имеет три версии конструктора, которые отличаются списком их параметров, следовательно:

|  |  |
| --- | --- |
| 1  2  3 | Circle()  Circle(**double** r)  Circle(**double** r, String c) |

В зависимости от фактического списка аргументов, используемых при вызове метода, будет вызван соответствующий конструктор. Если ваш список аргументов не соответствует ни одному из определенных методов, вы получите ошибку компиляции.

**10. Модификаторы контроля доступа- public или private.**

Контроль за доступом осуществляется с помощью модификатора, он может быть использован для управления видимостью класса или переменных –полей или методов внутри класса. Мы начнем со следующих двух модификаторов управления доступом:

public: класс / переменная / метод доступным и для всех других объектов в системе.

private: класс / переменная / метод доступным и в пределах только этого класса.

Например, в приведенном выше определении Circle, radius переменная-поле данных класса объявлена private. В результате, radius доступен внутри класса Circle , но не внутри класса TestCircle . Другими словами, вы не можете использовать "c1.radius" по отношению к радиусу C1 в классе TestCircle . Попробуйте вставить текст "System.out.println (c1.radius ) ;" в TestCircle и понаблюдать за сообщением об ошибке.

Попробуйте изменить radius на public, и повторно запустить пример.

С другой стороны, метод getRadius() определяется как public в классе Circle . Таким образом, он может быть вызван в классе TestCircle.

В нотации UML на диаграмме классов обозначается: общедоступные (public) элементы обозначены со знаком "+", в то время как закрытые (private) элементы со знаком " - ".

**11. Информация по сокрытию реализации и инкапсуляции.**

Класс инкапсулирует имя, статические атрибуты и динамическое поведение в "виде трех частей целого". Можно себе представить класс в виде некоторой коробки, на которой написано имя, внутри нее есть некоторое содержимое. После того, как класс определен, то есть вы туда сто-то положили, написали название на коробке, то можно закрыть "крышку" у этой коробки и поставить ее на полку. В дальнейшем как вы сами можете использовать эту коробку, так и можете предоставить возможность использовать ее всем желающим. Другими словами, любой желающий может открыть "крышку" этой коробки и использовать ее содержимое, то есть использовать ваш класс в своем приложении. Этого нельзя сделать, программируя в традиционном процедурном-ориентированном стиле, например на языке Cи, так как статические атрибуты (или переменные) разбросаны по всей программе и в файлах, заголовки которых вы включаете в код. Вы не сможете "вырезать" куски из части программы на Cи, включить их в другую программу и ожидать что такая программа запуститься без внесения в код серьезных изменений.

Переменные-поля данных класса, как правило, скрыты от внешнего слоя (то есть, недоступны другим классам), для этого осуществляется разграничение доступа или контроль доступа с использованием модификатора доступа - private. Доступ к закрытым переменным - полям класса предоставляются через методы, объявленные с модификатором public, например, getRadius () и getColor().

Использование такого подход соответствует принципу сокрытия информации – принципу инкапсуляции. То есть, объекты могут общаться друг с другом, только через хорошо определенные интерфейсы (публичные методы). Объектам не позволено знать детали реализации других объектов. Детали реализации всегда скрыты извне или инкапсулированы внутри класса. Сокрытие информации облегчает повторное использование класса.

Основное правило при создании определения классов: не объявляйте переменные общедоступными – с модификатором доступа public, если у вас на то нет веских оснований, не нарушайте принцип инкапсуляции.

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:* реализовать вывод информации из файла на экран

*Решение:*

*Листинг 1 – Функция main, запускающая исполнение программы (файл TestAll.java)*

|  |
| --- |
| package com.company;  public class TestAll {  public static void main(String[] args) {  Ball b1 = new Ball("Зелёный", "Футбольный", 200);  Ball b2 = new Ball("Жёлтый", "Регбистский", 1000);  b2.setColor("Оранжевый");  b1.outBall();  b2.outBall();  Dog d1 = new Dog("Батон", "OnePlus", 2);  Dog d2 = new Dog("Знахарь", "AfterSleep", 5);  d1.setAge(6);  d1.setPedigree("Monje");  d1.outDog();  d2.outDog();  Book bk1 = new Book(2037, "Клавелл", "Сёгун");  Book bk2 = new Book(305, "Глуховский", "Сумерки");  bk2.setPages(2005);  bk1.setPages(300);  bk1.outBook();  bk2.outBook();  }  } |

*Листинг 2 – Класс Ball (файл Ball.java)*

|  |
| --- |
| package com.company;  public class Ball {  private String color;  private String type;  private int price;  Ball(String color, String type, int price){  this.color = color;  this.type = type;  this.price = price;  }  void setColor(String color) {  this.color = color;  }  String getColor(){  return color;  }  void setType(String type){  this.type = type;  }  String getType(){  return type;  }  void setPrice(int pressure){  this.price = price;  }  int getPrice(){  return price;  }  public String toString() {  return this.color + ", price " + this.price;  }  public void outBall() {  System.out.println(color + " " + type + " ball costs " + price + " rub");  }  } |

*Листинг 3 – Класс Dog (файл Dog.java)*

|  |
| --- |
| package com.company;  public class Dog {  private String name;  private String pedigree;  private int age;  Dog(String name, String pedigree, int age){  this.name = name;  this.pedigree = pedigree;  this.age = age;  }  void setName(String name) {  this.name = name;  }  String getName(){  return name;  }  void setPedigree(String pedigree){  this.pedigree = pedigree;  }  String getPedigree(){  return pedigree;  }  void setAge(int age){  this.age = age;  }  int getAge(){  return age;  }  public String toString() {  return this.name + ", price " + this.age;  }  public void outDog() {  System.out.println(name + ", " + age + " лет, корм - " + pedigree);  }  } |

*Листинг 4 – Класс Book (файл Book.java)*

|  |
| --- |
| package com.company;  public class Book {  private int pages;  private String author;  private String title;  Book(int pages, String author, String title){  this.pages = pages;  this.author = author;  this.title = title;  }  void setPages(int pages) {  this.pages = pages;  }  int getPages(){  return pages;  }  void setAuthor(String author){  this.author = author;  }  String getAuthor(){  return author;  }  void setTitle(String title){  this.title = title;  }  String getTitle(){  return title;  }  public String toString() {  return this.pages + ", price " + this.title;  }  public void outBook() {  System.out.println("Книга " + title + ", автор - " + author + ", " + pages + "страниц");  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы были получены практические знания рыботы с классами Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 2

по дисциплине «Объектно-ориентированное программирование»

**«**ИСПОЛЬЗОВАНИЕ UML ДИАГРАММ В ОБЪЕКТНО- ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ**»**

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «13» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ РАБОТЫ**

Работа с UML-диаграммами классов.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Язык моделирования Unified Modeling Language (UML) является стандартом де-факто с 1998 года для проектирования и документирования объектно-ориентированных программ. Средствами UML в виде диаграмм можно графически изобразить класс и экземпляр класса.

Графически представляем класс в виде прямоугольника, разделенного на три области – область именования класса, область инкапсуляции данных и область операций (методы).

Имя (или сущность) : определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области). На рисунке 2.1 приведен общий вид UML диаграммы класса.

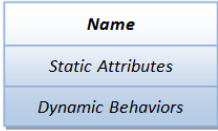


Рисунок 2.1 - Представление класса.

Методы (или поведение, функции, работа c данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“коробке” или прямоугольнике).

На рисунке 2.2 показано несколько примеров классов:

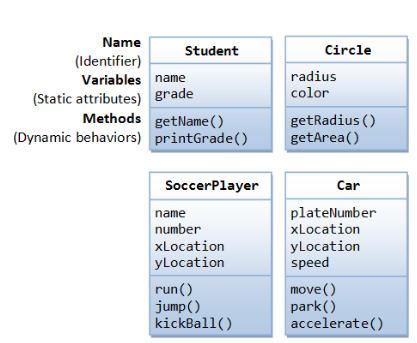


Рисунок 2.2 - Примеры экземпляров классов.

На рисунке 2.3 показаны два экземпляра класса типа Student "paul" и "peter".

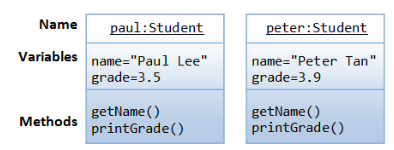


Рисунок 2.3 - Экземпляры класса Student.

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три области, одна содержит название, две вторых содержат переменные (поля данных класса) и методы класса, соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект класса) также представляется в виде прямоугольника, разделенного на три части, в первой части помещается надпись с именем экземпляра, например в instanceName:Classname и выделенная подчеркиванием ( название\_экземпляра : имя\_класса).

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:*

**Упражнение 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.

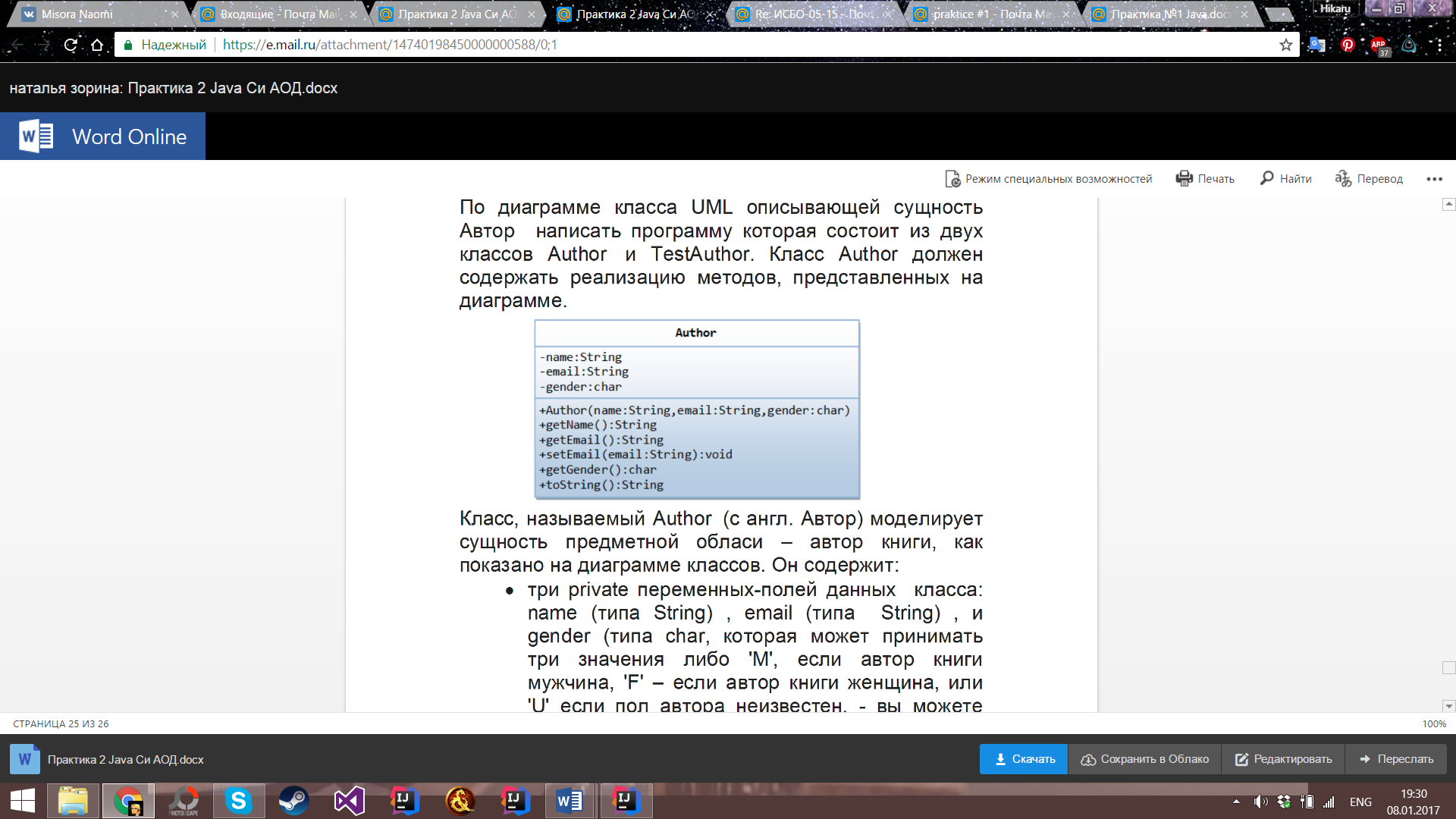


Рисунок 2.4 - Диаграмма класса Author.

**Упражнение 2.**

По UML диаграмме класса, представленной на рисунке 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.

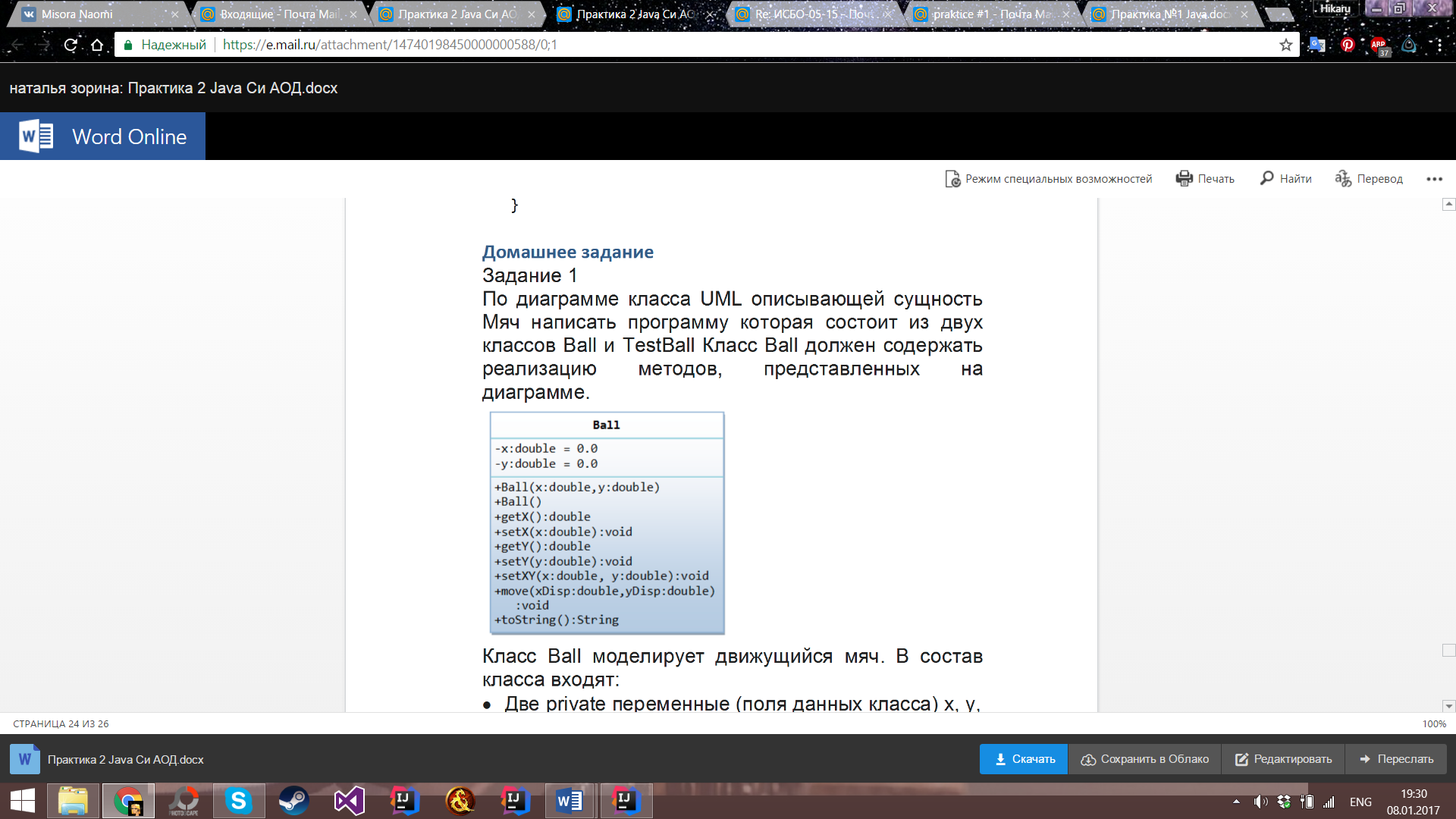


Рисунок 2.5 - Диаграмма класса Ball.

Класс Ball моделирует движущийся мяч. В состав класса входят:

Две переменные с модификатором private (поля данных класса): х, у, которые описывают положение мяча на поле.

Конструкторы, public методы получения и записи значений для private переменных.

Метод setXY (), который задает положение мяча и метод setXYSpeed(), чтобы задать скорость мяча

Метод move() , позволяет переместить мяч, так что что увеличивает х и у на данном участке на xDisp и yDisp, соответственно.

Метод toString(), который возвращает "Ball @ (х , у) " .

*Решение:*

**Упражнение 1.**

*Листинг 1 – Функция main, запускающая исполнение программы (файл TestAuthor.java)*

|  |
| --- |
| package com.company;  public class TestAuthor {  public static void main(String[] args){  Author author = new Author("Уильям Шекспир", " ", 'М');  author.setEmail("Shakespeare@mail.com");  System.out.println("Имя: " + author.getName() + ", Email: " + author.getEmail() + ", Пол: " + author.getGender());  }  } |

*Листинг 2 – Класс Author (файл Author.java)*

|  |
| --- |
| package com.company;  public class Author {  private String name;  private String email;  private char gender;  Author(String name, String email, char gender){  this.name = name;  this.email = email;  this.gender = gender;  }  String getName(){  return name;  }  String getEmail(){  return email;  }  void setEmail(String email){  this.email = email;  }  char getGender() {  return gender;  }  } |

**Упражнение 2.**

*Листинг 3 – Функция main, запускающая исполнение программы (файл TestBall.java)*

|  |
| --- |
| package com.company;  public class TestBall {  public static void main(String[] args) {  Ball ball = new Ball(100, 100);  System.out.println(ball);  ball.move(30, 15);  System.out.println(ball);  }  } |

*Листинг 4 – Класс Ball (файл Ball.java)*

|  |
| --- |
| package com.company;  public class Ball {  private double x = 0.0;  private double y = 0.0;  public Ball(){}  public Ball(double x, double y){  this.x = x;  this.y = y;  }  public double getX() {  return x;  }  public double getY() {  return y;  }  public void setX(double x) {  this.x = x;  }  public void setY(double y) {  this.y = y;  }  public void setXY(double x, double y){  this.x = x;  this.y = y; |

*Листинг 4 – Класс Ball (файл Ball.java) - продолжение*

|  |
| --- |
| }  public void move( double xDisp, double yDisp){  x+=xDisp;  y+=yDisp;  }  @Override  public String toString() {  return "Координаты мяча: ("+this.x+", "+this.y+").";  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы получены практические навыки работы с UML-диаграммами классови считывания с них данных для использования в программе.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 3

по дисциплине «Объектно-ориентированное программирование»

**«**НАСЛЕДОВАНИЕ. АБСТРАКТНЫЕ СУПЕРКЛАССЫ И ИХ ПОДКЛАССЫ В JAVA **»**

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «22» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ РАБОТЫ**

Освоить на практике работу с абстрактными классами и наследованием на Java.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

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.

|  |
| --- |
| **public** **abstract** **class** **Swim** {  // абстрактный метод плавать()  **abstract** **void** **neigh**();  // абстрактный класс может содержать и обычный метод  **void** **run**() {  System.out.println("Куда идешь?");  }  }  **class** **Swimmer** **extends** Swim {  ….  } |

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:*

**Упражнение 1.**

Создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы).

**Упражнение 2.**

Перепишите суперкласс Shape и его подклассы, так как это представлено на диаграмме Circle, Rectangle and Square.



Рисунок 3.1 – Диаграмма суперкласса Shape.

В этом задании, класс Shape определяется как абстрактный класс, который содержит:

* 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() .

**Упражнение 3.**

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.

Shape s1 = **new** Circle(**5.5**, "RED", **false**); // Upcast Circle to Shape

System.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());

// Take note that we downcast Shape s4 to Rectangle,

// which is a superclass of Square, instead of 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 to 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());

**Упражнение 4.**

Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

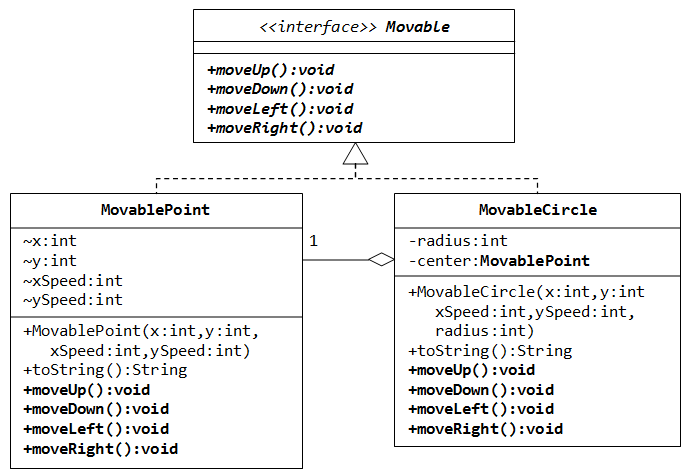


Рисунок 3.2. – Диаграмма реализации итерфейса Movable.

public interface Movable {

// saved as "Movable.java"

public void moveUp();

......

}

**Упражнение 5.**

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. Убедитесь, что две точки имеет одну и ту же скорость (нужен метод это проверяющий).

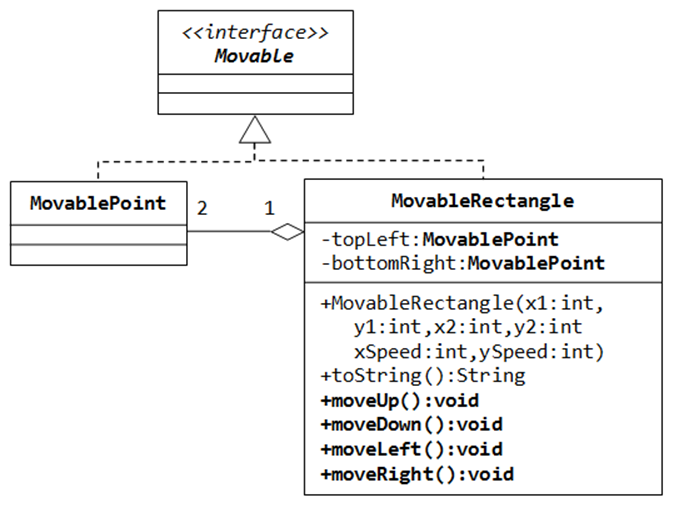


Рисунок 3.3 – Диаграмма класса MovableRectangle.

*Решение:*

**Упражнение 1.**

*Листинг 1 – Функция main, запускающая исполнение программы (файл Test.java)*

|  |
| --- |
| package com.company;  public class Test {  public static void main(String[] args){  Shape s1 = new Circle(5.5, "RED", false);  System.out.println(s1);  System.out.println(s1.getArea());  System.out.println(s1.getPerimeter());  System.out.println(s1.getColor());  System.out.println(s1.isFilled());  Circle c1 = (Circle)s1;  System.out.println(c1);  System.out.println(c1.getArea());  System.out.println(c1.getPerimeter());  System.out.println(c1.getColor());  System.out.println(c1.isFilled()); |

*Листинг 1 – Функция main, запускающая исполнение программы (файл Test.java) - продолжение*

|  |
| --- |
| System.out.println(c1.getRadius());  Shape s3 = new Rectangle(1.0, 2.0, "RED", false);  System.out.println(s3);  System.out.println(s3.getArea());  System.out.println(s3.getPerimeter());  System.out.println(s3.getColor());  Rectangle r1 = (Rectangle)s3;  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);  System.out.println(s4);  System.out.println(s4.getArea());  System.out.println(s4.getColor());  Rectangle r2 = (Rectangle)s4;  System.out.println(r2);  System.out.println(r2.getArea());  System.out.println(r2.getColor());  System.out.println(r2.getLength());  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());  }  } |

*Листинг 2 – Абстрактный класс Shape (файл Shape.java)*

|  |
| --- |
| package com.company;  public abstract class Shape {  protected String color;  protected boolean filled;  Shape(){  color = "";  filled = false;  }  Shape(String color, boolean filled){  this.color = color;  this.filled = filled;  }  String getColor(){  return color;  }  void setColor(String color){  this.color = color;  }  boolean isFilled(){  return filled;  }  void setFilled(boolean filled){  this.filled = filled;  }  abstract double getArea();  abstract double getPerimeter();  abstract String toJString*();*  *}* |

*Листинг 3 – Класс Square (файл Square.java)*

|  |
| --- |
| package com.company;  public class Square extends Rectangle{  protected double side;  Square(){  side = 0;  }  Square(double side){  this.side = side;  }  Square(double side, String color, boolean filled){  this.side = side;  this.color = color;  this.filled = filled;  }  double getSide(){  return side;  }  void setSide(double side){  this.side = side;  }  void setWidth(double side){  this.width = side;  }  void setLength(double side){  this.length = side;  }  String toJString(){  return ("Тип: квадрат, Сторона: " + getSide() + ", Площадь: " + getArea() + ", Периметр: " + getPerimeter());  }  } |

*Листинг 4 – Класс Rectangle (файл Rectangle.java)*

|  |
| --- |
| package com.company;  public class Rectangle extends Shape {  protected double width;  protected double length;  Rectangle(){  width = 0;  length = 0;  }  Rectangle(double width, double length){  this.width = width;  this.length = length;  }  Rectangle(double width, double length, String color, boolean filled){  this.width = width;  this.length = length;  this.color = color;  this.filled = filled;  }  double getWidth(){  return width;  }  void setWidth(double width) {  this.width = width;  }  double getLength(){  return length;  }  void setLength(double length){  this.length = length;  }  double getArea() {  return width\*length;  }  double getPerimeter(){  return 2\*(width+length);  }  String toJString(){  return ("Тип: прямоугольник, Ширина: " + getWidth() + ", Высота: " + getLength() +", Площадь: " + getArea() + ", Периметр: " + getPerimeter());  }  } |

*Листинг 5 – Класс Circle (файл Circle.java)*

|  |
| --- |
| package com.company;  public class Circle extends Shape{  protected double radius;  Circle(){  radius = 0;  }  Circle(double radius){  this.radius = radius;  }  Circle(double radius, String color, boolean filled){  super();  this.radius = radius;  }  double getRadius(){  return radius;  }  void setRadius(double radius) {  this.radius = radius;  }  double getArea(){  return Math.PI\*radius\*radius;  }  double getPerimeter(){  return 2\*Math.PI\*radius;  }  public String toJString(){  return ("Тип: окружность, Радиус: " + radius + ", Площадь: " + getArea() + ", Периметр: " + getPerimeter());  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы получены практические навыки работы с абстрактными классами Java.

Также был усвоен материал по наследованию классов в Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 4

по дисциплине «Объектно-ориентированное программирование»

**«**СОЗДАНИЕ GUI. СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ В JAVA **»**

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «25» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ РАБОТЫ**

Введение в событийное программирование на языке Java.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Данная практическая работа посвящена закреплению практических навыков по созданию приложений на Java c использованием следующих элементов GUI:

* Текстовые поля и области ввода текста;
* Менеджеры компоновки компонентов;
* Слушатель мыши;
* Создание меню.

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

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

Пример 1.

class LabExample extends JFrame

{

JTextField jta = new JTextField(10);

Font fnt = new Font("Times new roman",Font.BOLD,20);

LabExample()

{

super("Example");

setLayout(new FlowLayout());

setSize(250,100);

add(jta);

jta.setForeground(Color.PINK);

jta.setFont(fnt);

setVisible(true);

}

public static void main(String[]args)

{

new LabExample();

}

}

****

Рисунок 4.1

**Важная 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:

Пример 2.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class LabExample extends JFrame

{

JTextField jta1 = new JTextField(10);

JTextField jta2 = new JTextField(10);

JButton button = new JButton(" Add them up");

Font fnt = new Font("Times new roman",Font.BOLD,20);

LabExample()

{

super("Example");

setLayout(new FlowLayout());

setSize(250,150);

add(new JLabel("1st Number"));

add(jta1);

add(new JLabel("2nd Number"));

add(jta2);

add(button);

button.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent ae)

{

try

{

double x1 = Double.parseDouble(jta1.getText().trim());

double x2 = Double.parseDouble(jta2.getText().trim());

JOptionPane.showMessageDialog(null, "Result = "+(x1+x2),"Alert",JOptionPane.INFORMATION\_MESSAGE);

}

catch(Exception e)

{

JOptionPane.showMessageDialog(null, "Error in Numbers !","alert" , JOptionPane.ERROR\_MESSAGE);

}

}

});

setVisible(true);

}

public static void main(String[]args)

{

new LabExample();

}

}

**JTextArea**

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

Примeр 3.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class TextAreaExample extends JFrame

{

JTextArea jta1 = new JTextArea(10,25);

JButton button = new JButton("Add some Text");

public TextAreaExample()

{

super("Example");

setSize(300,300);

setLayout(new FlowLayout());

add(jta1);

add(button);

button.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent ae)

{

String txt = JOptionPane.showInputDialog(null,"Insert some text");

jta1.append(txt);

}

});

}

public static void main(String[]args)

{

new TextAreaExample().setVisible(true);

}

}

**Замечание.**

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

JTextArea txtArea = new JTextArea(20,20)

JScrollPane jScroll = new JScrollPane(txtArea);

// …

add(Scroll); // we add the scrollPane and not the text area.

Попробуйте выполнить сами!

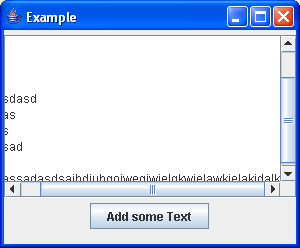


Рисунок 4.2

**Менеджеры компоновки компонентов или Layout Менеджеры.**

**Менеджер BorderLayout**:

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

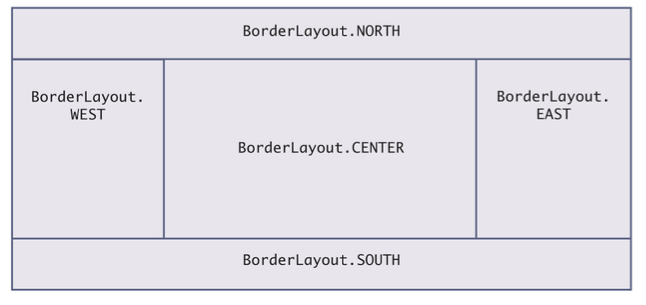


Рисунок 4.3

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

add(comp, BorderLayout.EAST);

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

**Менеджер GridLayout.**

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

Таблица 1.

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

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

Следующий пример иллюстрирует смесь компоновки различных компонентов.

Пример 4.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

class BorderExample extends JFrame

{

JPanel[] pnl = new JPanel[12];

public BorderExample()

{

setLayout(new GridLayout(3,4));

for(int i = 0 ; i < pnl.length ; i++)

{

int r = (int) (Math.random() \* 255);

int b = (int) (Math.random() \* 255);

int g = (int) (Math.random() \* 255);

pnl[i] = new JPanel();

pnl[i].setBackground(new Color(r,g,b));

add(pnl[i]);

}

pnl[4].setLayout(new BorderLayout());

pnl[4].add(new JButton("one"),BorderLayout.WEST);

pnl[4].add(new JButton("two"),BorderLayout.EAST);

pnl[4].add(new JButton("three"),BorderLayout.SOUTH);

pnl[4].add(new JButton("four"),BorderLayout.NORTH);

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

pnl[10].setLayout(new FlowLayout());

pnl[10].add(new JButton("one"));

pnl[10].add(new JButton("two"));

pnl[10].add(new JButton("three"));

pnl[10].add(new JButton("four"));

pnl[10].add(new JButton("fve"));

setSize(800,500);

}

public static void main(String[]args)

{

new BorderExample().setVisible(true);

}

}

Код представленный выше, будет иметь вид как на рисунке ниже.

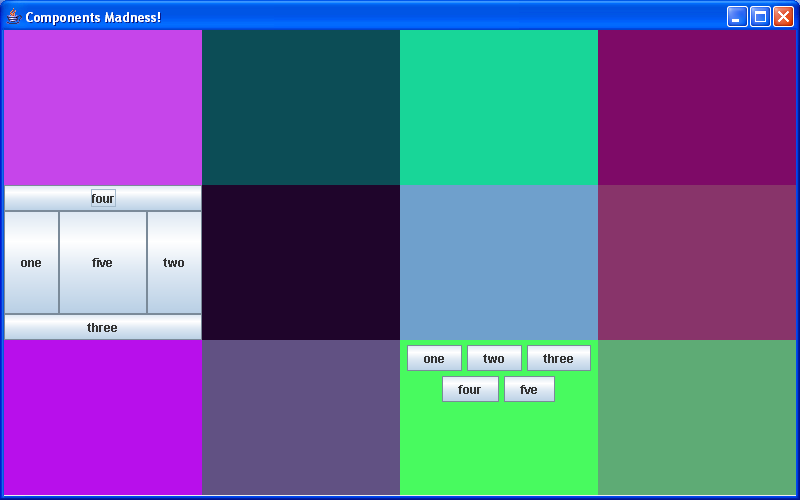


Рисунок 4.4

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

**Менеджер Null Layout Manager.**

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

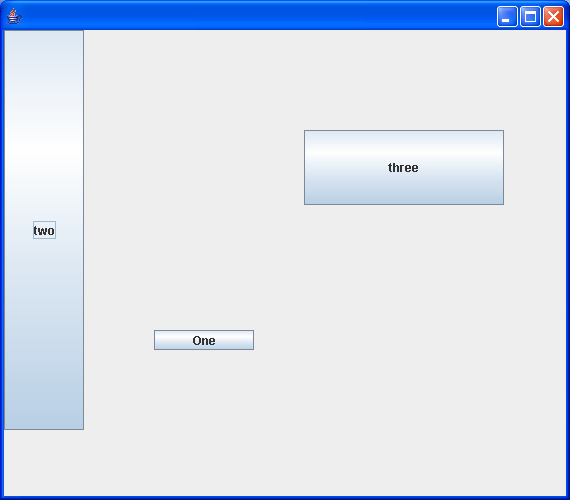


Рисунок 4.5

Пример 5.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

class NullLayout extends JFrame

{

JButton but1 = new JButton("One");;

JButton but2 = new JButton("two");;

JButton but3 = new JButton("three");;

public NullLayout()

{

setLayout(null);

but1.setBounds(150,300,100,20); // added at 150,300 width = 100, height=20

but2.setSize(80,400); // added at 0,0 width = 80, height=400

but3.setLocation(300,100);

but3.setSize(200,75);

// those two steps can be combined in one setBounds method call

add(but1);

add(but2);

add(but3);

setSize(500,500);

}

public static void main(String[]args)

{

new NullLayout().setVisible(true);

}

}

**Слушатель событий мыши MouseListener.**

Мы можем реализовывать слушателей мыши и также слушателей клавиатуры на компонентах GUI. Интерфейс MouseListener имеет следующие методы:

Таблица 1. Методы класса MouseListener.

|  |  |  |
| --- | --- | --- |
| Методы класса | | |
| Возвращаемое  значение | Прототип метода | Описание |
| void | [**mouseClicked**](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseListener.html#mouseClicked%28java.awt.event.MouseEvent%29)([MouseEvent](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseEvent.html) e) | Вызывается, когда кнопка мыши была нажата (нажата и отпущена) на области компонента. |
| void | [**mouseEntered**](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseListener.html#mouseEntered%28java.awt.event.MouseEvent%29)([MouseEvent](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseEvent.html) e) | Вызывается, когда мышь входит в область компонент. |
| void | [**mouseExited**](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseListener.html#mouseExited%28java.awt.event.MouseEvent%29)([MouseEvent](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseEvent.html) e) | Вызывается, когда мышь выходит из области компонента. |
| void | [**mousePressed**](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseListener.html#mousePressed%28java.awt.event.MouseEvent%29)([MouseEvent](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseEvent.html) e) | Вызывается при нажатии кнопки мыши на область компонента. |
| void | [**mouseReleased**](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseListener.html#mouseReleased%28java.awt.event.MouseEvent%29)([MouseEvent](file:///C:\jdk-1_5_0-doc\docs\api\java\awt\event\MouseEvent.html) e) | Вызывается, когда над областью компонента отпущена кнопка мыши. |

Слушатель мыMouseListener можно добавить к компоненту следующим образом:

Component.addMouseListener(listener);

Здесь слушатель является экземпляром класса, который реализует интерфейс MouseListener. Обратите внимание, что он должен обеспечивать выполнение всех методов, перечисленных в таблице 1 в данной практической работе.

Пример 6.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class MyMouse extends JFrame

{

JLabel lbl = new JLabel("");

public MyMouse()

{

super("Dude! Where's my mouse ?");

setSize(400,400);

setLayout(new BorderLayout());

add(lbl,BorderLayout.SOUTH);

addMouseListener(new MouseListener()

{

public void mouseExited(MouseEvent a){}

public void mouseClicked(MouseEvent a) {lbl.setText("X="+a.getX()+" Y="+a.getY());}

public void mouseEntered(MouseEvent a) {}

public void mouseReleased(MouseEvent a) {}

public void mousePressed(MouseEvent a) {}

});

}

public static void main(String[]args)

{

new MyMouse().setVisible(true);

}

}



Рисунок 4.6

**Создание меню.**

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

* JMenuBar: который представляет собой компонент, который содержит меню.
* JMenu: который представляет меню элементов для выбора.
* JMenuItem: представляет собой элемент, который можно кликнуть из меню.



Рисунок 4.7

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

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:*

**Упражнение 1.**

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

* одна кнопка JButton labeled “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.

*Решение:*

*Листинг 1 – Класс Window, с которого начинается исполнение программы*

*(файл Window.java)*

|  |
| --- |
| package com.company;  import javax.swing.\*;  import java.awt.\*;  import java.awt.event.\*;  import java.util.Random;  public class Window extends JFrame {  JButton firstButton = new JButton("AC Milan");  JButton secondButton = new JButton("Real Madrid");  JButton closeProgram = new JButton("Завершить");  JLabel result = new JLabel("Result: 0 X 0;");  JLabel lastScorer = new JLabel("Last Scorer: N/A;");  JLabel winner = new JLabel("Winner: DRAW;");  void setResult(int firstScore, int secondScore){  result.setText("Result: " + firstScore + " X " + secondScore + ";");  }  void setWinner(int firstScore, int secondScore){  if(firstScore > secondScore) winner.setText("Winner: AC Milan;");  else  if(firstScore < secondScore) winner.setText("Winner: Real Madrid;");  else  winner.setText("Winner: DRAW.");  }  int milanScores = 0;  int madridScores = 0;  Window() {  super("Window");  setLayout(new FlowLayout());  setSize(350, 200);  setVisible(true);  add(firstButton);  add(secondButton);  add(closeProgram);  add(result);  add(lastScorer);  add(winner); |

*Листинг 1 – Класс Window, с которого начинается исполнение программы*

*(файл Window.java) - продолжение*

|  |
| --- |
| firstButton.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  milanScores++;  lastScorer.setText("Last Scorer: AC Milan;");  setResult(milanScores, madridScores);  setWinner(milanScores, madridScores);  }  });  secondButton.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  madridScores++;  lastScorer.setText("Last Scorer: Real Madrid;");  setResult(milanScores, madridScores);  setWinner(milanScores, madridScores);  }  });  closeProgram.addMouseListener(new MouseAdapter() {  @Override  public void mouseClicked(MouseEvent e) {  JOptionPane.showMessageDialog(null, ("Match complete. \n" + result.getText() + "\n" + winner.getText() + "\n" + lastScorer.getText()));  System.exit(0);  }  });  }  public static void main(String[] args) {  new Window();  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы получены практические навыки работы с графическим интерфейсом в Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 5

по дисциплине «Объектно-ориентированное программирование»

**«**ПРОГРАММИРОВАНИЕ РЕКУРСИИ В JAVA **»**

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «26» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ РАБОТЫ**

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](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B0%D0%BB):

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… Чтобы вычислить 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](https://habrahabr.ru/post/261159/) и [Ханойская башня](https://habrahabr.ru/post/200758/)

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?

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:*

**Упражнение 1.**

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.

**Упражнение 2.**

Сумма цифр числа

Дано натуральное число N. Вычислите сумму его цифр.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).

**Упражнение 3.**

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.

*Решение:*

*Листинг 1 – Класс Main – первое упражнение (файл Main.java)*

|  |
| --- |
| package com.company;  import java.util.\*;  public class Main { |

*Листинг 1 – Класс Main – первое упражнение (файл Main.java) - продолжение*

|  |
| --- |
| public static void main(String[] args) {  System.out.println("Введите натуральные числа k и s:");  Scanner scanner = new Scanner(System.in);  int k = scanner.nextInt();  int s = scanner.nextInt();  System.out.println(recursion(0, 0, k, s)); // вызов рекурсивной функции  }  public static int recursion(int len, int sum, int k, int s) {  if (len == k) {  if (sum == s) {  return 1;  } else {  return 0;  }  }  int c = (len == 0 ? 1 : 0);  int res = 0;  for (int i = c; i < 10; i++) {  res += recursion(len + 1, sum + i, k, s);  }  return res;  }  } |

*Листинг 2 – Класс Task5 – второе упражнение (файл Task5.java)*

|  |
| --- |
| package com.company;  import java.util.\*;  public class Task5 {  public static void main(String[] args){  System.out.println("Введите число N:");  Scanner scanner = new Scanner(System.in);  int N = scanner.nextInt();  int res = 0;  System.out.println("Сумма цифр числа " + N + " равна " + sum(N, res));  }  public static int sum(int number, int res){  if(number == 0) return res;  res+=(number % 10);  number/=10;  return sum(number, res);  }  } |

*Листинг 3 – Класс Task6 – третье упражнение (файл Task6.java)*

|  |
| --- |
| package com.company;  import java.util.\*;  public class Task6 {  public static void main(String[] args) {  System.out.println("Введите число n:");  Scanner scanner = new Scanner(System.in);  int n = scanner.nextInt();  if (recursion(n, 2)) System.out.println("YES");  else System.out.println("NO");  }  public static boolean recursion(int n, int i) {  if (n < 2) {  return false;  }  else if (n == 2) {  return true;  }  else if (n % i == 0) {  return false;  }  else if (i < n / 2) {  return recursion(n, i + 1);  } else {  return true;  }  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы получены практические навыки работы с рекурсивными алгоритмами в Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 6

по дисциплине «Объектно-ориентированное программирование»

**«**ТЕХНИКИ СОРТИРОВКИ В JAVA **»**

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «27» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**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 надлежащее место 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 полиморфизма в сортировке.**

Техника программирования сортировок в Java 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.

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:*

**Упражнение 1.**

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

**Упражнение 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.

**Упражнение 3.**

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

*Решение:*

*Листинг 1 – Класс SortingStudentsByGPA (файл SortingStudentsByGPA.java)*

|  |
| --- |
| package com.company;  public class SortingStudentsByGPA {  static void outputArray(Student[] array) {  for (int i = 0; i < array.length; i++) {  System.out.println("Name: " + array[i].getName() + ", idNumber: " + array[i].getIdNumber() + ", GPA: " + array[i].getGPA());  }  }  static Student[] quickSort(Student[] array, int leftBorder, int rightBorder, GPAComparator cmp){  int leftMarker = leftBorder;  int rightMarker = rightBorder;  Student pivot = array[(leftMarker + rightMarker) / 2];  do {  while (cmp.compare(array[leftMarker], pivot) == 1) {  leftMarker++;  }  while (cmp.compare(array[rightMarker], pivot) == -1) {  rightMarker--;  }  if (leftMarker <= rightMarker) {  if (leftMarker < rightMarker) {  Student tmp = array[leftMarker];  array[leftMarker] = array[rightMarker];  array[rightMarker] = tmp;  }  leftMarker++;  rightMarker--;  }  } while (leftMarker <= rightMarker);  if (leftMarker < rightBorder) { |

*Листинг 1 – Класс SortingStudentsByGPA (файл SortingStudentsByGPA.java) - продолжение*

|  |
| --- |
| quickSort(array, leftMarker, rightBorder, cmp);  }  if (leftBorder < rightMarker) {  quickSort(array, leftBorder, rightMarker, cmp);  }  return array;  }  public static void main(String[] args){  Student[] studentTwo = new Student[5];  GPAComparator gpac = new GPAComparator();  studentTwo[0] = new Student("James", 5, 4.2);  studentTwo[1] = new Student("Neil", 9, 3.0);  studentTwo[2] = new Student("Ivan", 7, 3.5);  studentTwo[3] = new Student("Dan", 3, 4.9);  studentTwo[4] = new Student("Alex", 6, 3.4);  System.out.println("Сортировка по среднему баллу. \n");  System.out.println("Неотсортированный список: ");  outputArray(studentTwo);  System.out.println("\nОтсортированный список: ");  outputArray(quickSort(studentTwo, 0, studentTwo.length-1, gpac));  }  } |

*Листинг 2 – Интерфейс Comparator (файл Comparator.java)*

|  |
| --- |
| package com.company;  public interface Comparator {  int compare(Student a, Student b);  }; |

*Листинг 3 – Класс GPAComparator (файл GPAComparator.java)*

|  |
| --- |
| package com.company;  public class GPAComparator implements Comparator {  public int compare(Student a, Student b) {  if(a.getGPA() < b.getGPA())  return -1;  else  if(a.getGPA() == b.getGPA())  return 0;  else  return 1;  }  } |

*Листинг 4 – Класс Student (файл Student.java)*

|  |
| --- |
| package com.company;  public class Student {  private String name;  private int idNumber;  private double GPA;  Student(String name, int idNumber, double GPA){  this.name = name;  this.idNumber = idNumber;  this.GPA = GPA;  }  String getName(){  return name;  }  void setName(String name){  this.name = name;  }  int getIdNumber(){  return idNumber;  }  void setIdNumber(int idNumber){  this.idNumber = idNumber;  }  double getGPA(){ return GPA; }  void setGPA(double GPA){ this.GPA = GPA;}  } |

*Листинг 4 – Класс Test – второе упражнение (файл Test.java)*

|  |
| --- |
| package com.company;  public class Test {  static void outputArray(Student[] array) {  for (int i = 0; i < array.length; i++) {  System.out.println("Name: " + array[i].getName() + ", idNumber: " + array[i].getIdNumber() + ", GPA: " + array[i].getGPA());  } |

*Листинг 4 – Класс Test – второе упражнение (файл Test.java) - продолжение*

|  |
| --- |
| }  static Student[] insertionSort(Student[] array){  Student value;  for(int i = 1; i < array.length; i++){  value = array[i];  int j = i;  while(j > 0 && array[j - 1].getIdNumber() > value.getIdNumber()){  array[j] = array[j - 1];  j--;  }  array[j] = value;  }  return array;  }  public static void main(String[] args){  Student[] studentOne = new Student[5];  studentOne[0] = new Student("Peter", 0, 5.0);  studentOne[1] = new Student("Mark", 8, 4.4);  studentOne[2] = new Student("Sam", 2, 3.7);  studentOne[3] = new Student("Paul", 1, 3.9);  studentOne[4] = new Student("John", 4, 3.5);  System.out.println("Сортировка по id. \n");  System.out.println("Неотсортированный список: ");  outputArray(studentOne);  System.out.println("\nОтсортированный список: ");  outputArray(insertionSort(studentOne));}} |

*Листинг 5 – Класс TwoListsMergening (файл TwoListsMergening.java)*

|  |
| --- |
| package com.company;  public class TwoListsMerging {  static void outputArray(Student[] array) {  for (int i = 0; i < array.length; i++) {  System.out.println("Name: " + array[i].getName() + ", idNumber: " + array[i].getIdNumber() + ", GPA: " + array[i].getGPA());  }  }  static Student[] merge(Student[] first, Student[] second){  Student[] commonArray = new Student[first.length + second.length];  for (int i = 0; i < commonArray.length; i++){  if(i < first.length)  commonArray[i] = first[i];  else  commonArray[i] = second[i - first.length];}  return commonArray; |

*Листинг 5 – Класс TwoListsMergening (файл TwoListsMergening.java) - продолжение*

|  |
| --- |
| }  static Student[] mergeSort(Student[] source, int left, int right){  int delimiter = left + ((right - left) / 2) + 1;  if (delimiter > 0 && right > (left + 1)) {  mergeSort(source, left, delimiter - 1);  mergeSort(source, delimiter, right);  }  Student[] buffer = new Student[right - left + 1];  int cursor = left;  for (int i = 0; i < buffer.length; i++) {  if (delimiter > right || source[cursor].getIdNumber() > source[delimiter].getIdNumber()) {  buffer[i] = source[cursor];  cursor++;  } else {  buffer[i] = source[delimiter];  delimiter++;  }  }  System.arraycopy(buffer, 0, source, left, buffer.length);  return source;  }  public static void main(String[] args){  Student[] sortedById = new Student[5];  Student[] notQuiteSortedByGPA = new Student[5];  sortedById[0] = new Student("Peter", 0, 5.0);  sortedById[1] = new Student("Paul", 1, 3.9);  sortedById[2] = new Student("Sam", 2, 3.7);  sortedById[3] = new Student("John", 4, 3.5);  sortedById[4] = new Student("Mark", 8, 4.4);  notQuiteSortedByGPA[0] = new Student("Dan", 3, 4.9);  notQuiteSortedByGPA[1] = new Student("James", 6, 4.2);  notQuiteSortedByGPA[2] = new Student("Alex", 5, 3.4);  notQuiteSortedByGPA[3] = new Student("Ivan", 7, 3.5);  notQuiteSortedByGPA[4] = new Student("Neil", 9, 3.0);  outputArray(mergeSort(merge(sortedById, notQuiteSortedByGPA), 0, (sortedById.length + notQuiteSortedByGPA.length) - 1));  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы получены практические навыки работы с сортировочными алгоритмами на основе программирования в Java.

|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

Практическаяработа № 7

по дисциплине «Объектно-ориентированное программирование»

**«**ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ КОНТЕЙНЕРНЫХ КЛАССОВ ПРИ ПРОГРАММИРОВАНИИ НА JAVA **»**

Выполнил студент группы ИКБО-12-18 *Косогоров К.С.*

Принял Волков М.Ю.

Лабораторная работа выполнена «28» ноября 2019г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2019г.

Москва 2019

**ЦЕЛЬ РАБОТЫ**

Изучение на практике приемов работы со стандартными контейнерными классами Java Collection Framework.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Java Collections Framework — это набор связанных классов и интерфейсов, реализующих широко используемые структуры данных — коллекции. На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие в фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари).

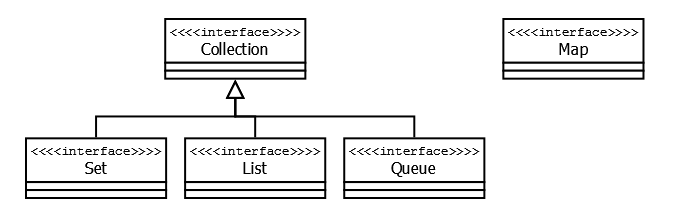


Рисунок 7.1

Vector — реализация динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Vector появился в JDK версии Java 1.0, но, как и Hashtable, эту коллекцию не рекомендуется использовать, если не требуется достижения потокобезопасности. Потому как в Vector, в отличии от других реализаций List, все операции с данными являются синхронизированными. В качестве альтернативы часто применяется аналог — ArrayList.

Stack — данная коллекция является расширением коллекции Vector. Была добавлена в Java 1.0 как реализация стека LIFO (last-in-first-out). Является частично синхронизированной коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

ArrayList — как и Vector является реализацией динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Как можно догадаться из названия, его реализация основана на обычном массиве. Данную реализацию следует применять, если в процессе работы с коллекцией предполагается частое обращение к элементам по индексу. Из-за особенностей реализации обращение к элементам по индексу, которое выполняется за константное время O(1). Использование данной коллекции рекомендуется избегать, если требуется частое удаление/добавление элементов в середине коллекции.

LinkedList — вид реализации List. Позволяет хранить любые данные, включая null. Данная коллекция реализована на основе двунаправленного связного списка (каждый элемент списка имеет ссылки на предыдущий и следующий). Добавление и удаление элемента из середины, доступ по индексу, значению происходит за линейное время O(n), а из начала и конца за константное время O(1). Ввиду реализации, данную коллекцию можно использовать как абстрактный тип данных — стек или очередь. Для этого в ней реализованы соответствующие методы.

Интерфейс Set.

Представляет собой неупорядоченную коллекцию, которая не может содержать одинаковые элементы и является программной моделью математического понятия «множество».

Интерфейс Queue.

Этот интерфейс описывает коллекции с предопределённым способом вставки и извлечения элементов, а именно — очереди FIFO (first-in-first-out). Помимо методов, определённых в интерфейсе Collection, определяет дополнительные методы для извлечения и добавления элементов в очередь.

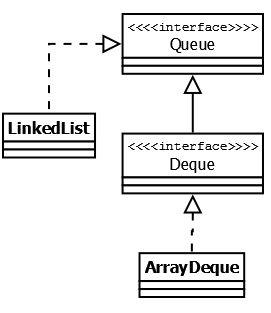


Рисунок 7.2

PriorityQueue — является единственной прямой реализацией интерфейса Queue (была добавлена, как и интерфейс Queue, в Java 1.5), не считая класса LinkedList, который так же реализует этот интерфейс, но был реализован намного раньше. Особенностью данной очереди является возможность управления порядком элементов. По умолчанию, элементы сортируются с использованием «natural ordering», но это поведение может быть переопределено при помощи объекта Comparator, который задаётся при создании очереди. Данная коллекция не поддерживает null в качестве элементов.

ArrayDeque — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out). Интерфейс Deque и реализация ArrayDeque были добавлены в Java 1.6. Эта коллекция представляет собой реализацию с использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null.

Как заявлено в документации, эта коллекция работает быстрее чем Stack, если используется как LIFO коллекция, а также быстрее чем LinkedList, если используется как FIFO коллекция.

**ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ:**

*Задание:*

Напишите программу в виде консольного приложения, которая моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую; карта «0» побеждает карту «9».

Карточная игра “ В пьяницу”. В этой игре карточная колода раздается поровну двум игрокам. Далее они открывают по одной верхней карте, и тот, чья карта старше, забирает себе обе открытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт, - проигрывает.

Для простоты будем считать, что все карты различны по номиналу и что самая младшая карта побеждает самую старшую карту (“шестерка берет туз”).

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

**Входные данные.**

Программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая - 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

**Выходные данные.**

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа должна вывести слово botva.

**Упражнение 1.**

Используйте для организации хранения структуру данных Stack.

**Упражнение 2.**

Используйте для организации хранения структуру данных Queue.

**Упражнение 3.**

Используйте для организации хранения структуру данных Dequeue.

**Упражнение 3.**

Используйте для организации хранения структуру данных DoubleList.

*Решение:*

*Листинг 1 – Класс GameOnDequeue (файл GameOnDequeue.java)*

|  |
| --- |
| package com.company;  import java.util.ArrayDeque;  import java.util.Deque;  public class GameOnDequeue {  static void outputDeque(Deque deque){  for (Object element : deque) {  System.out.print(element + " ");  }  }  static String determineWinner(Deque first, Deque second, int turns){  if(turns != 106) {  if(!first.isEmpty())  return ("first " + turns);  else  return ("second " + turns);  }  else  return "botva";  }  public static void main(String[] args) {  Deque<Integer> firstPlayer = new ArrayDeque<>();  firstPlayer.add(3);  firstPlayer.add(6);  firstPlayer.add(0);  firstPlayer.add(2);  firstPlayer.add(7);  Deque<Integer> secondPlayer = new ArrayDeque<>();  secondPlayer.add(9);  secondPlayer.add(4); |

*Листинг 1 – Класс GameOnDequeue (файл GameOnDequeue.java) - продолжение*

|  |
| --- |
| secondPlayer.add(8);  secondPlayer.add(1);  secondPlayer.add(5);  System.out.print("Игрок 1: ");  outputDeque(firstPlayer);  System.out.print("\nИгрок 2: ");  outputDeque(secondPlayer);  System.out.println();  System.out.println();  int countOfTurns = 0;  while(!firstPlayer.isEmpty() && !secondPlayer.isEmpty()){  if(firstPlayer.peek() > secondPlayer.peek()){  firstPlayer.addLast(firstPlayer.remove());  firstPlayer.addLast(secondPlayer.remove());  }  else {  secondPlayer.addLast(secondPlayer.remove());  secondPlayer.addLast(firstPlayer.remove());  }  countOfTurns++;  System.out.println("Ход №" + countOfTurns);  outputDeque(firstPlayer);  System.out.println();  outputDeque(secondPlayer);  System.out.println();  System.out.println();  }  System.out.println(determineWinner(firstPlayer, secondPlayer, countOfTurns));  }  } |

*Листинг 2 –Класс GameOnDoubleList (файл GameOnDoubleList.java)*

|  |
| --- |
| package com.company;  import java.util.LinkedList;  public class GameOnDoubleList {  static void outputList(LinkedList list){  for (Object element : list) {  System.out.print(element + " ");  }  }  static String determineWinner(LinkedList first, LinkedList second, int turns){  if(turns != 106) {  if(!first.isEmpty())  return ("first " + turns);  else  return ("second " + turns);  }  else  return "botva";  }  public static void main(String[] args) {  LinkedList<Integer> firstPlayer = new LinkedList<>();  firstPlayer.add(3);  firstPlayer.add(6);  firstPlayer.add(0);  firstPlayer.add(2);  firstPlayer.add(7);  LinkedList<Integer> secondPlayer = new LinkedList<>();  secondPlayer.add(9);  secondPlayer.add(4);  secondPlayer.add(8);  secondPlayer.add(1);  secondPlayer.add(5);  System.out.print("Игрок 1: ");  outputList(firstPlayer);  System.out.print("\nИгрок 2: ");  outputList(secondPlayer);  System.out.println();  System.out.println(); |

*Листинг 2 – Класс GameOnDoubleList (файл GameOnDoubleList.java) - продолжение*

|  |
| --- |
| int countOfTurns = 0;  while(!firstPlayer.isEmpty() && !secondPlayer.isEmpty()){  if(firstPlayer.getFirst() > secondPlayer.getFirst()){  firstPlayer.addLast(firstPlayer.remove());  firstPlayer.addLast(secondPlayer.remove());  }  else {  secondPlayer.addLast(secondPlayer.remove());  secondPlayer.addLast(firstPlayer.remove());  }  countOfTurns++;  System.out.println("Ход №" + countOfTurns);  outputList(firstPlayer);  System.out.println();  outputList(secondPlayer);  System.out.println();  System.out.println();  }  System.out.println(determineWinner(firstPlayer, secondPlayer, countOfTurns));  }  } |

*Листинг 3 – Класс GameOnQueue (файл GameOnQueue.java)*

|  |
| --- |
| package com.company;  import java.util.ArrayDeque;  import java.util.Queue;  public class GameOnQueue {  static void outputQueue(Queue queue){  for (Object element : queue) {  System.out.print(element + " ");  }  }  static String determineWinner(Queue first, Queue second, int turns){  if(turns != 106) {  if(!first.isEmpty())  return ("first " + turns);  else  return ("second " + turns);  }  else  return "botva";  }  public static void main(String[] args) {  Queue<Integer> firstPlayer = new ArrayDeque<>();  firstPlayer.add(3);  firstPlayer.add(6);  firstPlayer.add(0);  firstPlayer.add(2);  firstPlayer.add(7);  Queue<Integer> secondPlayer = new ArrayDeque<>();  secondPlayer.add(9);  secondPlayer.add(4);  secondPlayer.add(8);  secondPlayer.add(1);  secondPlayer.add(5);  System.out.print("Игрок 1: ");  outputQueue(firstPlayer);  System.out.print("\nИгрок 2: ");  outputQueue(secondPlayer);  System.out.println(); |

*Листинг 3 – Класс GameOnQueue (файл GameOnQueue.java) - продолжение*

|  |
| --- |
| System.out.println();  int countOfTurns = 0;  while(!firstPlayer.isEmpty() && !secondPlayer.isEmpty()){  if(firstPlayer.peek() > secondPlayer.peek()){  firstPlayer.add(firstPlayer.remove());  firstPlayer.add(secondPlayer.remove());  }  else {  secondPlayer.add(secondPlayer.remove());  secondPlayer.add(firstPlayer.remove());  }  countOfTurns++;  System.out.println("Ход №" + countOfTurns);  outputQueue(firstPlayer);  System.out.println();  outputQueue(secondPlayer);  System.out.println();  System.out.println();  }  System.out.println(determineWinner(firstPlayer, secondPlayer, countOfTurns));  }  } |

*Листинг 4 – Класс GameOnStack (файл GameOnStack.java)*

|  |
| --- |
| package com.company;  import java.util.Stack;  public class GameOnStack {  static void outputDeck(Stack stack){  Stack<Object> temp = new Stack<>();  for (Object element: stack)  temp.push(element);  while(!temp.isEmpty()){  System.out.print(temp.pop() + " ");  }  }  static String determineWinner(Stack first, Stack second, int turns){  if(turns != 106) {  if(!first.isEmpty())  return ("first " + turns);  else  return ("second " + turns);  }  else  return "botva";  }  public static void main(String[] args) {  Stack<Integer> firstPlayer = new Stack<>();  firstPlayer.push(7);  firstPlayer.push(2);  firstPlayer.push(0);  firstPlayer.push(6);  firstPlayer.push(3);  Stack<Integer> secondPlayer = new Stack<>();  secondPlayer.push(5);  secondPlayer.push(1);  secondPlayer.push(8);  secondPlayer.push(4);  secondPlayer.push(9);  Stack<Integer> tempStack = new Stack<>();  int summaryNumberOfElements = firstPlayer.size() + secondPlayer.size();  int countOfTurns = 0; |

*Листинг 4 – Класс GameOnStack (файл GameOnStack.java) - продолжение*

|  |
| --- |
| while(firstPlayer.size() != summaryNumberOfElements && secondPlayer.size() != summaryNumberOfElements){  if(firstPlayer.peek() > secondPlayer.peek()){  int temp = firstPlayer.pop();  while(!firstPlayer.isEmpty())  tempStack.push(firstPlayer.pop());  tempStack.push(temp);  tempStack.push(secondPlayer.pop());  while(!tempStack.isEmpty())  firstPlayer.push(tempStack.pop());  }  else{  int temp = secondPlayer.pop();  while(!secondPlayer.isEmpty())  tempStack.push(secondPlayer.pop());  tempStack.push(temp);  tempStack.push(firstPlayer.pop());  while(!tempStack.isEmpty())  secondPlayer.push(tempStack.pop());  }  countOfTurns++;  System.out.println("Ход №" + countOfTurns);  outputDeck(firstPlayer);  System.out.println();  outputDeck(secondPlayer);  System.out.println();  System.out.println();  }  System.out.println(determineWinner(firstPlayer, secondPlayer, countOfTurns));  }  } |

**ВЫВОДЫ ПО РАБОТЕ:**

В ходе выполнения работы получены практические навыки работы со стандартными контейнерными классами Java Collectiob Framework.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

# РТУ МИРЭА

**Институт информационных технологий**

## Кафедра ИППО (инструментального и прикладного программного обеспечения)

## Направление подготовки 09.03.04 Программная инженерия

**Разработка программных продуктов и проектирование**

Профиль **информационных систем**

**Рабочая тетрадь**

**по практическим занятиям и лабораторным работам**

**Дисциплина: «Объектно-ориентированное программирование»**

**Студент \_\_\_\_\_\_\_\_\_\_**

**Группа \_\_\_\_\_\_\_\_\_\_\_**

**Преподаватель: Торхов А.Е.**

Москва 2019

## 1. Цели освоения дисциплины

Дисциплина «Объектно-ориентированное программирование» имеет своей целью способствовать формированию у обучающихся профессиональных (ПК-13) компетенций в соответствии с требованиями ФГОС ВО по направлению подготовки бакалавров 09.03.04 «Программная инженерия» с учетом специфики профиля подготовки – «Разработка программных продуктов и проектирование информационных систем».

## 2. Место дисциплины в структуре основной профессиональной образовательной программы

Дисциплина «Объектно-ориентированное программирование» является обязательной дисциплиной базовой части блока «Дисциплины» учебного плана направления подготовки бакалавров 09.03.04 «Программная инженерия» с профилем подготовки «Разработка программных продуктов и проектирование информационных систем».

Данная дисциплина изучается в течение двух семестров (второго и третьего) на I и

II курсе. Общая трудоемкость дисциплины составляет 10 зачетных единиц (360 акад. час.) Формы промежуточной аттестации: экзамен и курсовая работа в первом семестре изучения дисциплины, зачет во втором семестре изучения дисциплины.

## Практические занятия

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | № раздела дисциплины согласно РП  (*место и дата проведения*  *ПрЗ*) | Тематика практических занятий | Трудоемкость  (в часах).  *Отметка о выполнении* |
| 1 | *1,2,3,4 ауд. Г-226-1 дата*  *«07» сентября 2019г* | Практическая работа №1  Классы, как новые типы данных.  Поля данных и методы. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 2 | *2,3,7 ауд. Г-226-1 дата*  *«14»сентября 2019г* | Практическая работа №2 Использование UML диаграмм в объектно-ориентированном программирование | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 3 | *9,11*  *ауд. Г-226-1 дата*  *«21»сентября*  *2019г* | Практическая работа №3 Наследование. Абстрактные суперклассы и их подклассы в Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | *9,11*  *ауд. Г-226-1 дата*  *«28»сентября 2019г* | Практическая работа №4 Создание GUI. Событийное программирование в Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 5 | *11*  *ауд. Г-226-1 дата*  *«05»октября 2019г* | Практическая работа №5  Программирование рекурсии в Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 6 | *11*  *ауд. Г-226-1 дата*  *«12»октября 2019г* | Практическая работа №6  Техники сортировки Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 7 | *11*  *ауд. Г-226-1 дата*  *«19»октября 2019г* | Практическая работа №7 Использование стандартных контейнерных классов при программирование на Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 8 | *11*  *ауд. Г-226-1 дата*  *«26»октября 2019г* | Практическая работа №8  Generics | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 9 | *13*  *ауд. Г-226-1 дата*  *«02»ноября 2019г* | Практическая работа №9 Создание пользовательских исключений. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_» Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | *12*  *ауд. Г-226-1 дата*  *«09»ноября 2019г* | Практическая работа №10  Паттерны – абстрактная фабрика. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 11 | *12*  *ауд. Г-226-1 дата*  *«16»ноября 2019г* | Практическая работа №11 АТД. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 12 | *12*  *ауд. Г-226-1 дата*  *«23»ноября 2019г* | Практическая работа №12 Работа со строками. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 13 | *10*  *ауд. Г-226-1 дата*  *«30»ноября 2019г* | Практическая работа №13 Работа с датой и времени. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 14 | *10*  *ауд. Г-226-1 дата*  *«07»декабря 2019г* | Практическая работа №14 Хэш таблица. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_» Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 15 | *16*  *ауд. Г-226-1 дата*  *«14»декабря 2019г* | Практическая работа №15 Хэш карта. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 16 | *16*  *ауд. Г-226-1 дата*  *«21»декабря 2019г* | Практическая работа №16 Итоговая работа. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |

**Лабораторные работы**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | № раздела дисциплины согласно РП  (*место и дата проведения*) | Тематика лабораторных работ | Трудоемкость  (в часах).  *Отметка о выполнении* |
| 1 | *1,2*  *ауд. Г-226-1 дата*  *«07»сентября 2019г* | Лабораторная работа №1. Циклы, условия, переменные и массивы в Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 2 | *2,3,4*  *ауд. Г-226-1 дата*  *«14»сентября 2019г* | Лабораторная работа №2.  ООП в Java. Понятие класса. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора))* |
| 3 | *2,3,4*  *ауд. Г-226-1 дата*  *«21»сентября 2019г* | Лабораторная работа №3. Наследование в Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 4 | *11*  *ауд. Г-226-1 дата*  *«28»сентября 2019г* | Лабораторная работа №4. Интерфейсы в Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | *11*  *ауд. Г-226-1 дата*  *«05»октября 2019г* | Лабораторная работа №5.  Cоздание программ с графическим интерфейсом пользователя на Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 6 | *10,11 ауд. Г-226-1 дата*  *«12»октября 2019г* | Лабораторная работа №6. Обработка событий в Java программах с графическим интерфейсом пользователя. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 7 | *10,11 ауд. Г-226-1 дата*  *«19»октября 2019г* | Лабораторная работа №7.  Коллекции, очереди, списки Java. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 8 | *10,11 ауд. Г-226-1 дата*  *«26»ноября 2019г* | Лабораторная работа №8. Работа с файлами. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 9 | *10,11 ауд. Г-226-1 дата*  *«02»ноября 2019г* | Лабораторная работа №9.  Работа с исключениями. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 10 | *10,11 ауд. Г-226-1 дата*  *«09»ноября 2019г* | Лабораторная работа №10. Работа с дженериками. Стирание типов. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора))* |
| 11 | *5*  *ауд. Г-226-1 дата*  *«16»ноября 2019г* | Лабораторная работа №10. Работа с дженериками. Стирание типов. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 12 | *5*  *ауд. Г-226-1 дата*  *«23»ноября 2019г* | Лабораторная работа №12. Обработка строк. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_» Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя (тьютора)* |
| 13 | *10*  *ауд. Г-226-1 дата*  *«30»ноября 2019г* | Лабораторная работа №12. Использование регулярных выражений в Java-приложениях. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 14 | *10*  *ауд. Г-226-1 дата*  *«07»декабря 2019г* | Лабораторная работа №14. Хэш таблица. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 15 | *14,15 ауд. Г-226-1 дата*  *«14»декабря 2019г* | Лабораторная работа №15.  Хэш карта. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |
| 16 | *14,15 ауд. Г-226-1 дата*  *«21»декабря 2019г* | Лабораторная работа №16. Итоговая работа. | *2 часа*  *«\_\_\_\_\_\_\_\_\_\_\_\_»*  *Дата «21»декабря 2019г подпись студента \_\_\_\_\_\_ Дата «21»декабря 2019г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *ф.и.о. преподавателя*  *(тьютора)* |

*В результате обучения по дисциплине «Объектно-ориентированное программирование» освоены следующие компетенции:*

*- готовность к использованию методов и инструментальных средств исследования объектов профессиональной деятельности*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Подпись ФИО*

**Используемая литература:**

а) основная литература:

1. Мейер, Б. Основы объектно-ориентированного программирования [Электронный ресурс] : учебник / Б. Мейер. — Электрон. дан. — Москва : , 2016. — 969 с. — Режим доступа: https://e.lanbook.com/book/100306. — Загл.

с экрана.

б) дополнительная литература:

1. Буч, Г. Язык UML. Руководство пользователя. [Электронный ресурс] / Г. Буч, Д. Рамбо, И. Якобсон. — Электрон. дан. — М. : ДМК Пресс, 2008. — 496 с. — Режим доступа: http://e.lanbook.com/book/1246 — Загл. с экрана.

в) учебно-методические пособия:

1. Романенко, В.В. Объектно-ориентированное программирование [Электронный ресурс] : учебное пособие / В.В. Романенко. — Электрон.

дан. — Москва : ТУСУР, 2014. — 475 с. — Режим доступа:

https://e.lanbook.com/book/110354. — Загл. с экрана.

1. Бурмистров, А.В. Программирования на языке JAVA. Методические указания к лабораторным работам. [Электронный ресурс] : Учебнометодические пособия — Электрон. дан. — Пенза : ПензГТУ, 2014. — 150 с. — электронный ресурс: <http://e.lanbook.com/book/62752>

г) ресурсы сети интернет:

1. http://www.oracle.com/technetwork/java/index.html - Технология Java
2. <http://www.ibm.com/developerworks/ru/edu/j-intserv/index.html>
3. http://www.ibm.com/developerworks/ru/edu/ws-jax/index.html
4. https://docs.oracle.com/javase/tutorial/

1. Unified Modeling Language – унифицированный язык моделирования, UML. [↑](#footnote-ref-1)