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

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

Практические РАБОТы

по дисциплине «Программирование на языка Джава»

Выполнил студент группы ИКБО-02-21 Семянников Н.С.

Принял старший преподаватель Рачков А.В.

Практические работы работа выполнены «\_\_\_»\_\_\_\_\_\_\_2022г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2022г.

Москва 2022

Оглавление

[Практическая работа № 1 3](#_Toc119042213)

[Практическая работа № 2 8](#_Toc119042214)

[Практическая работа № 3 12](#_Toc119042215)

[Практическая работа № 4 22](#_Toc119042216)

[Практическая работа № 5 25](#_Toc119042217)

[Практическая работа № 6 29](#_Toc119042218)

[Практическая работа № 7 36](#_Toc119042219)

[Практическая работа № 8 42](#_Toc119042220)

[Практическая работа № 9 46](#_Toc119042221)

[Практическая работа № 10 48](#_Toc119042222)

[Практическая работа № 11 53](#_Toc119042223)

[Практическая работа № 12 63](#_Toc119042224)

[Практическая работа № 13 67](#_Toc119042225)

[Практическая работа № 14 69](#_Toc119042226)

[Практическая работа № 15 72](#_Toc119042227)

[Практическая работа № 16 78](#_Toc119042228)

[Практическая работа № 17 87](#_Toc119042229)

[Практическая работа № 18 91](#_Toc119042230)

[Практическая работа № 19 93](#_Toc119042231)

[Практическая работа № 20 96](#_Toc119042232)

[Практическая работа № 21 105](#_Toc119042233)

[Практическая работа № 22 108](#_Toc119042234)

[Практическая работа № 23 112](#_Toc119042235)

[Практическая работа № 24 113](#_Toc119042236)

# Практическая работа № 1

## Цель работы

Изучить работу с классами в Java

## Теоретическое введение

Класс 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.

Переменные: содержит статические атрибуты класса

Методы: описывают динамическое поведение класса.

Переменные и методы, входящие в состав класса, формально называется переменные-поля данных класса и методы 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.

Конструктор – это специальный метод класса, который используется для создания и инициализации всех переменных-полей данных класса.

Перегрузка методов означает, что несколько методов могут иметь то же самое имя метод, но сами методы могут иметь различные реализации. Тем не менее, различные реализации должны быть различимы по списку их аргументов

Контроль за доступом осуществляется с помощью модификатора, он может быть использован для управления видимостью класса или переменных – полей или методов внутри класса.

* public: класс / переменная / метод доступными для всех других объектов в системе
* private: класс / переменная / метод доступными в пределах только этого класса

## Выполнение лабораторной работы

### Задание:

Необходимо реализовать простейший класс на языке программирования Java. Не забудьте добавить метод toString() к вашему классу. Так-же в программе необходимо предусмотреть класс-тестер для тестирования класса и вывода информации об объекте.

### Решение:

#### Класс «Собака»

Листинг 1 - Код "Собака"

Класс принимает и возвращает кличку и возраст собаки

|  |
| --- |
| public class Dog {  private String name;  private int age;  public Dog(String n, int a) {  name = n;  age = a;  }  public Dog(String n) {  name = n;  age = 0;  }  public Dog() {  name = "Pup";  age = 0;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public String toString() {  return this.name + ", age " + this.age;  }  public void intoHumanAge() {  System.out.println(name + "'s age in human years is " + age \* 7 + " years");  }  } |

#### Класс-тестер «Собака»

Листинг 2 - Код тестер "Собака"

Проверка работы класса

|  |
| --- |
| public class TestDog {  public static void main(String[] args) {  Dog d1 = new Dog("Mike", 3);  Dog d2 = new Dog("Helen", 7);  Dog d3 = new Dog("Bob");  d3.setAge(1);  System.out.println(d1);  d1.intoHumanAge();  d2.intoHumanAge();  d3.intoHumanAge();  }  } |

#### Результат тестирования класса «Собака» представлен на рисунке 1

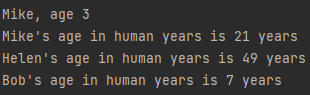


Рисунок 1 - Результат тестирования класса «Собака»

#### Класс «Мяч»

Листинг 3 - Код "Мяч"

Класс принимает радиус и цвет круга, а возвращает его площадь

|  |
| --- |
| public class Circle {  private double radius;  private String color;  public Circle(double r, String cl) {  radius = r;  color = cl;  }  public Circle(double r) {  radius = r;  color = "Red";  }  public Circle() {  radius = 5;  color = "Red";  }  public double getRadius() {  return radius;  }  public void setRadius(double r) {  radius = r;  }  public String getColor() {  return color;  }  public void setColor(String cl) {  color = cl;  }  public String toString() {  return radius + " - " + color;  }  public void getArea() {  System.out.println(Math.PI \* Math.pow(radius, 2));  }  } |

#### Класс-тестер «Мяч»

Листинг 4 - Код тестер "Мяч"

Проверка работы класса

|  |
| --- |
| public class TestCircle {  public static void main(String[] args) {  Circle c1 = new Circle(3, "Green");  Circle c2 = new Circle(8);  Circle c3 = new Circle();  c1.setColor("Blue");  System.out.println(c1);  System.out.println(c2);  System.out.println(c3);  c2.getArea();  }  } |

#### Результат тестирования класса «Мяч» представлен на рисунке 2

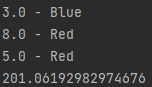


Рисунок 2 - Результат тестирования класса «Мяч»

#### Класс «Книга»

Листинг 5 - Код "Книга"

Класс принимает количество страниц и название книги, а возвращает примерный вес книги

|  |
| --- |
| public class Book {  private int pages;  private String name;  public Book(int p, String n) {  pages = p;  name = n;  }  public Book(int p) {  pages = p;  name = "Lion The Fat";  }  public Book() {  pages = 200;  name = "Lion The Fat";  }  public int getPages() {  return pages;  }  public void setPages(int p) {  pages = p;  }  public String getName() {  return name;  }  public void setName(String n) {  name = n;  }  public String toString() {  return pages + " - " + name;  }  public void getWeight() {  System.out.println(pages \* 1.5 + " grams");  }  } |

#### Класс-тестер «Книга»

Листинг 6 - Код тестер "Книга"

Проверка работы класса

|  |
| --- |
| public class TestBook {  public static void main(String[] args) {  Book b1 = new Book(500, "Slim Lion");  Book b2 = new Book(800);  Book b3 = new Book();  b2.setName("Happy Man");  System.out.println(b1);  System.out.println(b2);  System.out.println(b3);  b2.getWeight();  }  } |

#### Результат тестирования класса «Книга» представлен на рисунке 3

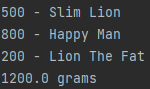


Рисунок 3 - Результат тестирования класса «Книга»

## Выводы по работе:

Изучил инкапсуляцию и работу с классами в Java.

# Практическая работа № 2

## Цель работы

Изучить обозначения и работу с UML диаграммами.

## Теоретическое введение

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.

Имя: определяет класс.

Переменные: содержит статические атрибуты класса, или описывают свойства класса

Методы: описывают динамическое поведение класса.

На рисунке 4 приведен общий вид UML диаграммы класса.



Рисунок 4 – UML

## Выполнение лабораторной работы

### Задание:

Необходимо написать программу, которая состоит из двух классов Author и TestAuthor.

Написать программу, которая состоит из двух классов. Один из них Ball должен реализовывать сущность мяч, а другой с названием TestBall тестировать работу созданного класса.

### Решение:

#### Класс «Автор»

Листинг 7 - Код "Автор"

Класс принимает имя, почту и пол автора, возвращает полученные данные

|  |
| --- |
| package T1;  public class Author {  private String name, email;  private char gender;  public Author(String name, String email, char gender) {  this.name = name;  this.email = email;  if (gender != 'm' && gender != 'f')  gender = 'u';  this.gender = gender;  }  public String getName() {  return name;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public char getGender() {  return gender;  }  public String toString() {  return name + "(" + (gender == 'f' ? "ms" : gender) + ") at " + email;  }  } |

#### Класс-тестер «Автор»

Листинг 8 - Код тестер "Автор"

Проверка работы класса

|  |
| --- |
| package T1;  public class TestAuthor {  public static void main(String[] args) {  Author a1 = new Author("Ivan Popov", "ivan@mail.ru", 'm');  System.out.println(a1.getName());  System.out.println(a1.getEmail());  System.out.println(a1.getGender());  a1.setEmail("ivanov@yandex.ru");  System.out.println(a1);  }  } |

#### Результат тестирования класса «Автор» представлен на рисунке 5

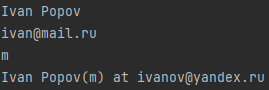


Рисунок 5 - Результат тестирования класса «Автор»

#### Класс «Мяч»

Листинг 9 - Код "Мяч"

Класс принимает координаты мячика, а также сдвигает его на определённое расстояние

|  |
| --- |
| package T2;  public class Ball {  private double x = 0.0, y = 0.0;  public Ball(double x, double y) {  this.x = x;  this.y = y;  }  public Ball() {  x = 0;  y = 0;  }  public double getX() {  return x;  }  public void setX(double x) {  this.x = x;  }  public double getY() {  return y;  }  public void setY(double y) {  this.y = y;  }  public void setXY(double x, double y) {  this.x = x;  this.y = y;  }  public void move(double xDisp, double yDisp) {  x += xDisp;  y += yDisp;  }  public String toString() {  return "Lab2.T2.Ball @ (" + x + ", " + y + ")";  }  } |

#### Класс-тестер «Мяч»

Листинг 10 - Код тестер "Мяч"

Проверка работы класса

|  |
| --- |
| package T2;  public class TestBall {  public static void main(String[] args) {  Ball b = new Ball(3,0);  System.out.println(b);  b.move(10, 7);  System.out.println(b);  }  } |

#### Результат тестирования класса «Мяч» представлен на рисунке 6



Рисунок 6 - Результат тестирования класса «Мяч»

## Выводы по работе:

Научился писать классы по UML диаграммам.

# Практическая работа № 3

## Цель работы

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

## Теоретическое введение

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.

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.

## Выполнение лабораторной работы

### Задание:

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

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

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.

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

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).

### Решение:

#### Класс «Shape»

Листинг 11 - Код "Shape"

Абстрактный класс объявляет стандартные значения цвета и заполненности для всех элементов, которые его наследуют, а также обязывает использовать методы getArea, getPerimeter и toString

|  |
| --- |
| package T1\_2;  public abstract class Shape {  protected String color;  protected boolean filled;  public Shape() {  color = "Red";  filled = true;  }  public String getColor() {  return color;  }  public void setColor(String color) {  this.color = color;  }  public boolean isFilled() {  return filled;  }  public void setFilled(boolean filled) {  this.filled = filled;  }  public abstract double getArea();  public abstract double getPerimeter();  public abstract String toString();  } |

#### Класс «Circle»

Листинг 12 - Код "Circle"

Класс принимает радиус окружности и возвращает его периметр и площадь

|  |
| --- |
| package T1\_2;  public class Circle extends Shape {  protected double radius;  public Circle() {  radius = 5;  }  public Circle(double r) {  radius = r;  }  public Circle(double r, String cl) {  radius = r;  color = cl;  }  public Circle(double r, String cl, boolean fl) {  radius = r;  color = cl;  filled = fl;  }  public double getRadius() {  return radius;  }  public void setRadius(int radius) {  this.radius = radius;  }  @Override  public double getArea() {  return Math.PI \* radius \* radius;  }  @Override  public double getPerimeter() {  return 2 \* Math.PI \* radius;  }  @Override  public String toString() {  return "Circle {" +  "radius=" + radius +  ", color='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

#### Класс «Rectangle»

Листинг 13 - Код "Rectangle"

Класс принимает размеры прямоугольника и возвращает его площадь и периметр

|  |
| --- |
| package T1\_2;  public class Rectangle extends Shape {  protected double width, height;  public Rectangle() {  width = 10;  height = 10;  }  public Rectangle(double w) {  width = w;  height = 10;  }  public Rectangle(double w, double h) {  width = w;  height = h;  }  public Rectangle(double w, double h, String cl) {  width = w;  height = h;  color = cl;  }  public Rectangle(double w, double h, String cl, boolean fl) {  width = w;  height = h;  color = cl;  filled = fl;  }  public double getHeight() {  return height;  }  public double getWidth() {  return width;  }  public void setHeight(double height) {  this.height = height;  }  public void setWidth(double width) {  this.width = width;  }  @Override  public double getPerimeter() {  return 2 \* (width + height);  }  @Override  public double getArea() {  return width \* height;  }  @Override  public String toString() {  return "Rectangle {" +  "width=" + width +  ", height=" + height +  ", color='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

#### Класс «Square»

Листинг 14 - Код "Square"

Класс идентичен классу Rectangle, но принимает только размеры одной из сторон

|  |
| --- |
| package T1\_2;  public class Square extends Rectangle {  public Square() {  width = 5;  height = 5;  }  public Square(double sd) {  width = sd;  height = sd;  }  public Square(double sd, String cl) {  color = cl;  width = sd;  height = sd;  }  public Square(double sd, String cl, boolean fl) {  color = cl;  filled = fl;  width = sd;  height = sd;  }  public double getSide() {  return width;  }  public void setSide(double sd) {  width = sd;  height = sd;  }  @Override  public void setWidth(double width) {  super.width = width;  }  @Override  public void setHeight(double height) {  super.height = height;  }  @Override  public String toString() {  return "Square {" +  ", width=" + width +  ", height=" + height +  ", color='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

#### Класс «TestAbstract»

Листинг 15 - Код "TestAbstract"

Проверка работы классов

|  |
| --- |
| package T1\_2;  public class TestAbstract {  public static void main(String[] args) {  Circle c = new Circle();  Rectangle re = new Rectangle();  Square sq = new Square();  System.out.println(c);  System.out.println("Area: " + c.getArea());  System.out.println("Perimeter: " + c.getPerimeter());  System.out.println("Radius: " + c.getRadius());  System.out.println(re);  System.out.println("Area: " + re.getArea());  System.out.println("Perimeter: " + re.getPerimeter());  System.out.println("Width: " + re.getWidth());  System.out.println(sq);  System.out.println("Area: " + sq.getArea());  System.out.println("Perimeter: " + sq.getPerimeter());  System.out.println("Side: " + sq.getSide());  }  } |

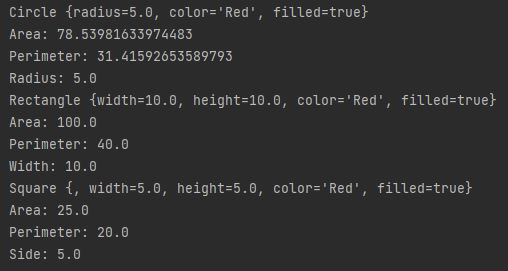


Рисунок 7 - Результат "TestAbstract"

#### Класс «Task3»

Листинг 16 - Код "Task3"

Проверка на ошибки

|  |
| --- |
| package T3;  import T1\_2.Circle;  import T1\_2.Rectangle;  import T1\_2.Shape;  import T1\_2.Square;  public class Task\_3 {  public static void main(String[] args) {  //Преобразование подкласса в родительский класс  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());  //Нет доступа к методу внутри класса Circle, так как создан объект класса Shape  //System.out.println(s1.getRadius());  //Создаём новый объект класса Circle и, меняя класс у переменной s1, присваиваем её новому объекту  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());  System.out.println(c1.getRadius()); //Теперь радиус выводится  //Shape s2 = new Shape(); //Невозможно создать абстрактный класс  //Преобразование подкласса в родительский класс  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());  //System.out.println(s3.getHeight()); //Нет доступа к методу подкласса  Rectangle r1 = (Rectangle)s3; //Преобразуем родительский класс в подкласс  System.out.println(r1);  System.out.println(r1.getArea());  System.out.println(r1.getColor());  System.out.println(r1.getHeight()); //Появился доступ к методу подкласса  Shape s4 = new Square(6.6); //Преобразование под-подкласса в родительский класс  System.out.println(s4);  System.out.println(s4.getArea());  System.out.println(s4.getColor());  //System.out.println(s4.getSide()); //Нет доступа к методу подкласса  Rectangle r2 = (Rectangle)s4; //Преобразуем родительский класс в подкласс  System.out.println(r2);  System.out.println(r2.getArea());  System.out.println(r2.getColor());  //System.out.println(r2.getSide()); //Доступа к методу класса Square до сих пор нет!  System.out.println(r2.getHeight());  Square sq1 = (Square)r2; //Преобразуем класса Rectangle в Square  System.out.println(sq1);  System.out.println(sq1.getArea());  System.out.println(sq1.getColor());  System.out.println(sq1.getSide()); //Появился доступ к методу подкласса  System.out.println(sq1.getHeight()); //Остался доступ к методу род. класса  }  } |

#### Класс «Movable»

Интерфейс обязывает использовать методы, которые прописаны внутри него

|  |
| --- |
| package T4;  public interface Movable {  public void moveUp();  public void moveDown();  public void moveLeft();  public void moveRight();  } |

#### Класс «MovableCircle»

Листинг 17 - Код "MovableCircle"

Класс реализует интерфейс «Movable», создаёт элемент класс MovablePoint в указанной координате, а затем сдвигает предоставленный круг на определённую дистанцию

|  |
| --- |
| package T4;  public class MovableCircle implements Movable {  private int radius;  private MovablePoint center;  public MovableCircle(int x, int y, int xSpeed, int ySpeed) {  center = new MovablePoint(x, y, xSpeed, ySpeed);  }  @Override  public String toString() {  return "MovableCircle {" +  "radius=" + radius +  ", center=" + center +  '}';  }  @Override  public void moveUp() {  center.y += center.ySpeed;  }  @Override  public void moveDown() {  center.y -= center.ySpeed;  }  @Override  public void moveLeft() {  center.x -= center.xSpeed;  }  @Override  public void moveRight() {  center.x += center.xSpeed;  }  } |

#### Класс «MovablePoint»

Листинг 18 - Код "MovablePoint"

Класс для создания точки окружности/точки

|  |
| --- |
| package T4;  public class MovablePoint implements Movable {  protected int x, y, xSpeed, ySpeed;  public MovablePoint(int x, int y, int xSpeed, int ySpeed) {  this.x = x;  this.y = y;  this.xSpeed = xSpeed;  this.ySpeed = ySpeed;  }  @Override  public void moveUp() {  y += ySpeed;  }  @Override  public void moveDown() {  y -= ySpeed;  }  @Override  public void moveLeft() {  x -= xSpeed;  }  @Override  public void moveRight() {  x += xSpeed;  }  @Override  public String toString() {  return "MovablePoint {" +  "x=" + x +  ", y=" + y +  ", xSpeed=" + xSpeed +  ", ySpeed=" + ySpeed +  '}';  }  } |

#### Класс «TestMovable»

Листинг 19 - Код "TestMovable"

Проверка работы программы

|  |
| --- |
| package T4;  public class TestMovable {  public static void main(String[] args) {  MovablePoint mv = new MovablePoint(10, 0, 3, 7);  MovableCircle mc = new MovableCircle(15, 20, 10, 40);  System.out.println(mv.toString());  System.out.println(mc.toString());  mc.moveUp();  mc.moveRight();  System.out.println(mc);  }  } |

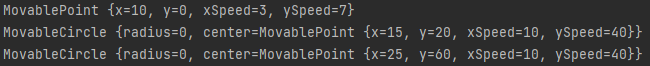


Рисунок 8 - Результат "TestMovable"

#### Класс «MovableRectangle»

Листинг 20 - Код "MovableRectangle"

Класс выполняет те же действия, что и MovableCircle, но используется две точки (верхняя левая и нижняя правая)

|  |
| --- |
| package T5;  import T4.Movable;  import T4.MovablePoint;  public class MovableRectangle implements Movable {  private MovablePoint tl, br;  public MovableRectangle(int x1, int y1, int x2, int y2, int xSpeed, int ySpeed) {  tl = new MovablePoint(x1, y1, xSpeed, ySpeed);  br = new MovablePoint(x2, y2, xSpeed, ySpeed);  }  @Override  public void moveUp() {  tl.moveUp();  br.moveUp();  }  @Override  public void moveDown() {  tl.moveDown();  br.moveDown();  }  @Override  public void moveLeft() {  tl.moveLeft();  br.moveLeft();  }  @Override  public void moveRight() {  tl.moveRight();  br.moveRight();  }  @Override  public String toString() {  return "MovableRectangle {" +  "tl=" + tl +  ", br=" + br +  '}';  }  } |

#### Класс «TestRect»

Листинг 21 - Код "TestRect"

Проверка работы класса

|  |
| --- |
| package T5;  public class TestRect {  public static void main(String[] args) {  MovableRectangle mr = new MovableRectangle(5, 10, 10, 5, 10, 10);  System.out.println(mr);  mr.moveUp();  mr.moveRight();  System.out.println(mr);  }  } |



Рисунок 9 - Результат "TestRect"

## Выводы по работе:

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

# Практическая работа № 4

## Цель работы

Введение в событийное программирование на языке Java.

## Теоретическое введение

TextField - текстовое поле или поля для ввода текста (можно ввести только одну строку).

Компонент TextArea похож на TextField, но в него можно вводить более одной строки.

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

### Менеджер BorderLayout:

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

### Менеджер GridLayout:

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

### 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)).

### Слушатель событий мыши MouseListener.

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

* mouseClicked
* mouseEntered
* mouseExited
* mousePressed
* mouseReleased

## Выполнение лабораторной работы

### Задание:

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид.

### Решение:

#### Класс «MyMouse»

Листинг 22 - Код "MyMouse"

Создание формы

|  |
| --- |
| import com.sun.tools.jconsole.JConsoleContext;  import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  import javax.swing.border.Border;  class MyMouse extends JFrame  {  JButton AC = new JButton("AC Milan");  JButton Real = new JButton("Real Madrid");  JLabel Res=new JLabel("Result: 0 X 0");  JLabel Last=new JLabel("Last Scorer: N/A");  JLabel Winner=new JLabel("Winner: DRAW");  public int AC\_int=0;  public int Real\_int=0;  Border border = BorderFactory.createLineBorder(Color.BLACK, 4);  public class TestActionListener implements ActionListener  {  public void actionPerformed(ActionEvent e)  {  AC\_int++;  Res.setText("Result: "+AC\_int+" X "+Real\_int);  Last.setText("Last Scorer: AC Milan");  Winner.setText("Winner: "+((AC\_int>Real\_int)?"AC Milan":((AC\_int==Real\_int)?"DRAW":"Real Madrid")));  }  }  public class TestActionListener1 implements ActionListener  {  public void actionPerformed(ActionEvent e)  {  Real\_int++;  Res.setText("Result: "+AC\_int+" X "+Real\_int);  Last.setText("Last Scorer: Real Madrid");  Winner.setText("Winner: "+((AC\_int>Real\_int)?"AC Milan":((AC\_int==Real\_int)?"DRAW":"Real Madrid")));  }  }  public MyMouse()  {  setLayout(null);  setSize(400,400);  AC.setBounds(50,80,400,60);  ActionListener actionListener = new TestActionListener();  AC.addActionListener(actionListener);  add(AC);  Real.setBounds(460,80,400,60);  ActionListener actionListener1 = new TestActionListener1();  Real.addActionListener(actionListener1);  add(Real);  Res.setBounds(50,30,810,50);  Res.setBorder(border);  Res.setFont(new Font("Serif", Font.PLAIN, 18));  Res.setHorizontalAlignment(JLabel.CENTER);  add(Res);  Last.setBounds(50,150,400,50);  Last.setBorder(border);  Last.setFont(new Font("Serif", Font.PLAIN, 18));  Last.setHorizontalAlignment(JLabel.CENTER);  add(Last);  Winner.setBounds(460,150,400,50);  Winner.setBorder(border);  Winner.setFont(new Font("Serif", Font.PLAIN, 18));  Winner.setHorizontalAlignment(JLabel.CENTER);  add(Winner);  setSize(930, 400);  setLocationRelativeTo(null);  }  public static void main(String[]args)  {  new MyMouse().setVisible(true);  }  } |

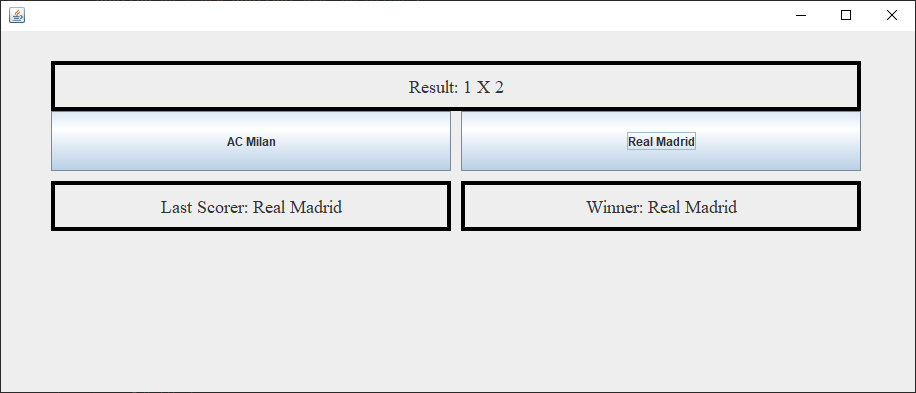


Рисунок 10 - Результат

## Выводы по работе:

Изучил работу с GUI и создание GUI элементов.

# Практическая работа № 5

## Цель работы

Разработка и программирование рекурсивных алгоритмов на языке Java.

## Теоретическое введение

В контексте языка программирования рекурсия — это некий активный метод (или подпрограмма) вызываемый сам по себе непосредственно, или вызываемой другим методом (или подпрограммой) косвенно.

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.

## Выполнение лабораторной работы

### Задание:

7. Разложение на множители

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

8. Палиндром

Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO. При решении этой задачи нельзя пользоваться циклами, в решениях на питоне нельзя использовать срезы с шагом, отличным от 1.

10. Разворот числа

Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке. При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика. Функция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

### Решение:

#### Класс «PrimeFactors»

Листинг 23 - Код "PrimeFactors"

Программа принимает число, затем рекурсивно разбирает все множители этого числа с помощью метода GetPrimes и проверяет является ли множитель простым числом с помощью метода isPrime

|  |
| --- |
| import java.util.Scanner;  public class PrimeFactors {  public static int GetPrimes(int n, int i) {  try {  if (n % i == 0) {  if (isPrime(i, 2)) {  System.out.println(i);  }  }  if (i <= n) {  return GetPrimes(n, i + 1);  } else {  return 1;  }  } catch (java.lang.StackOverflowError e) {  return 0;  }  }  public static boolean isPrime(int n, int i) {  if (n <= 2)  return (n == 2);  if (n % i == 0)  return false;  if (i \* i > n)  return true;  return isPrime(n, i + 1);  }  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n;  System.out.print("Введите число (n > 2): ");  n = sc.nextInt();  if (n > 1) {  System.out.println("Простые числа: ");  if (GetPrimes(n, 2) == 0) {  System.out.println("Стак переполнился");  }  } else {  System.out.println("Число меньше 2");  }  }  } |

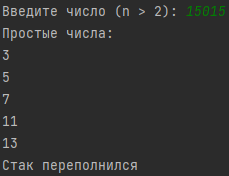


Рисунок 11 - Результат

#### Класс «Palindrome»

Листинг 24 - Код "Palindrome"

Программа принимает строку на вход, затем рекурсивно сверяет начальный и конечный символ, сдвигая начало и конец к середине строки. Если метод дошёл до середины, значит это палиндром

|  |
| --- |
| import java.util.Scanner;  public class Palindrome {  public static boolean isPalindrome(String s, int len, int i) {  if (i >= len / 2)  return true;  if (s.charAt(i) != s.charAt(len - i - 1))  return false;  return isPalindrome(s, len, i + 1);  }  public static void main(String[] args) {  String s;  Scanner sc = new Scanner(System.in);  System.out.print("Введите строку: ");  s = sc.next();  if (isPalindrome(s.toLowerCase(), s.length(), 0)) {  System.out.println("YES");  } else {  System.out.println("NO");  }  }  } |



Рисунок 12 - Результат если слово палиндром



Рисунок 13 - Результат если не палиндром

#### Класс «MirrorNumber»

Листинг 25 - Код "MirrorNumber"

Программа принимает число на вход, затем в методе MirrorNumber рекурсивно делит число на 10 для того, чтобы перейти к следующей цифре числа и записывает остаток к отзеркаленному числу

|  |
| --- |
| import java.util.Scanner;  public class MirrorNumber {  public static int mirrorNumb(int n, int a) {  if (n == 0) {  return a;  }  return mirrorNumb(n / 10, a \* 10 + n % 10);  }  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  n = mirrorNumb(n, 0);  System.out.println(n);  }  } |



Рисунок 14 - Результат

## Выводы по работе:

Изучил работу рекурсии на языке Java.

# Практическая работа № 6

## Цель работы

Освоение на практике методов сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

## Теоретическое введение

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

### 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.

### Алгоритм быстрой сортировки

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

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.

## 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:

#### Класс «Student»

Листинг 26 - Код "Student"

Класс принимает id, оценку, фамилию и имя студента

|  |
| --- |
| import java.util.Comparator;  public class Student {  protected int id, mark;  protected String surname, name;  public int getId() {  return id;  }  public String getName() {  return name;  }  public String getSurname() {  return surname;  }  public int getMark() {  return mark;  }  public void setName(String name) {  this.name = name;  }  public void setSurname(String surname) {  this.surname = surname;  }  public void setMark(int mark) {  this.mark = mark;  }  public void setId(int id) {  this.id = id;  }  public Student(int id, int mark, String surname, String name) {  this.id = id;  this.mark = mark;  this.surname = surname;  this.name = name;  }  @Override  public String toString() {  return "Student{" +  "id = " + id +  ", mark = " + mark +  ", surname = '" + surname + '\'' +  ", name = '" + name + '\'' +  '}';  }  } |

#### Класс «SortingStudentsByGPA»

Листинг 27 - Код "SortingStudentsByGPA"

Класс реализует интерфейс «Comparator» для того, чтобы сверять оценки студентов методом быстрой сортировки

|  |
| --- |
| import java.util.Comparator;  public class SortingStudentsByGPA implements Comparator<Student> {  @Override  public int compare(Student s1, Student s2) {  return s1.mark - s2.mark;  }  } |

Класс «Sorts»

Листинг 28 - Код "Sorts"

Программа сортирует студентов с помощью следующих сортировок:

Сортировка вставками: метод «insertionSort»

Быстрая сортировка: метод «QuickSort»

Сортировка слиянием: методы «mergeSort», «merge» и «mergeArrays»

|  |
| --- |
| import java.util.Arrays;  import java.util.Random;  import java.util.Scanner;  public class Sorts extends SortingStudentsByGPA {  public static void insertionSort(Student[] stud, int n) {  Student tmp;  for (int i = 1; i < n; i++) {  for (int j = i; j > 0 && stud[j - 1].getId() > stud[j].getId(); j--) {  tmp = stud[j];  stud[j] = stud[j - 1];  stud[j - 1] = tmp;  }  }  }  public static void QuickSort(Student[] stud, int n) {  for (int i = 0; i < n - 1; i++) {  System.out.println(new SortingStudentsByGPA().compare(stud[i], stud[i + 1]));  }  }  public static void createArray(Student[] stud, int n) {  Random r = new Random();  int[] ids = new int[n];  int tmp, randPos;  for (int i = 0; i < n; i++) {  ids[i] = i;  }  for (int i = 0; i < n; i++) {  randPos = r.nextInt(ids.length);  tmp = ids[i];  ids[i] = ids[randPos];  ids[randPos] = tmp;  }  for (int i = 0; i < n; i++) {  stud[i] = new Student(ids[i], r.nextInt(1, 5), "Smith", "John");  }  }  public static void mergeSort(Student[] st, int n) {  if (n < 2) {  return;  }  int mid = n / 2;  Student[] l = new Student[mid];  Student[] r = new Student[n - mid];  for (int i = 0; i < mid; i++) {  l[i] = st[i];  }  for (int i = mid; i < n; i++) {  r[i - mid] = st[i];  }  mergeSort(l, mid);  mergeSort(r, n - mid);  merge(st, l, r, mid, n - mid);  }  public static void merge(Student[] a, Student[] l, Student[] r, int left, int right) {  int i = 0, j = 0, k = 0;  while (i < left && j < right) {  if (l[i].getId() <= r[j].getId()) {  a[k++] = l[i++];  } else {  a[k++] = r[j++];  }  }  while (i < left) {  a[k++] = l[i++];  }  while (j < right) {  a[k++] = r[j++];  }  }  public static void mergeArrays(Student[] st1, Student[] st2, Student[] res, int n) {  for (int i = 0; i < n \* 2; i++) {  if (i < n)  res[i] = st1[i];  else  res[i] = st2[i - n];  }  }  public static void displayArray(Student[] stud, int n) {  for (int i = 0; i < n; i++) {  System.out.println(stud[i].toString());  }  }  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n;  System.out.print("Array size: ");  n = sc.nextInt();  Student[] students = new Student[n];  createArray(students, n);  displayArray(students, n);  System.out.println("\nInsertion sort");  insertionSort(students, n);  displayArray(students, n);  System.out.println("\n");  students = new Student[n];  createArray(students, n);  displayArray(students, n);  System.out.println("\nQuick sort");  Arrays.sort(students, new SortingStudentsByGPA());  displayArray(students, n);  n /= 2;  students = new Student[n];  Student[] students1 = new Student[n];  createArray(students, n);  createArray(students1, n);  Student[] result = new Student[n \* 2];  mergeArrays(students, students1, result, n);  System.out.println("\n");  n \*= 2;  displayArray(result, n);  mergeSort(result, n);  System.out.println("\nMerge sort");  displayArray(result, n);  }  } |

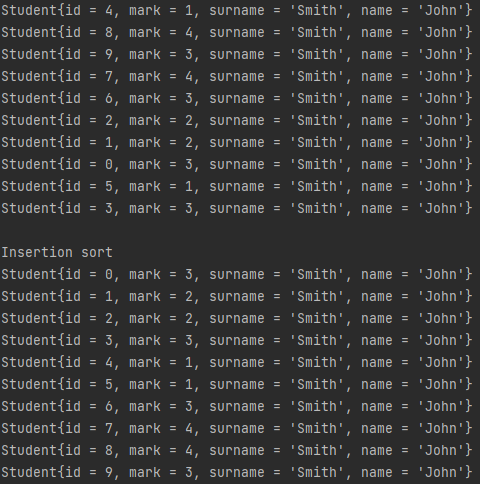


Рисунок 15 - Сортировка вставками

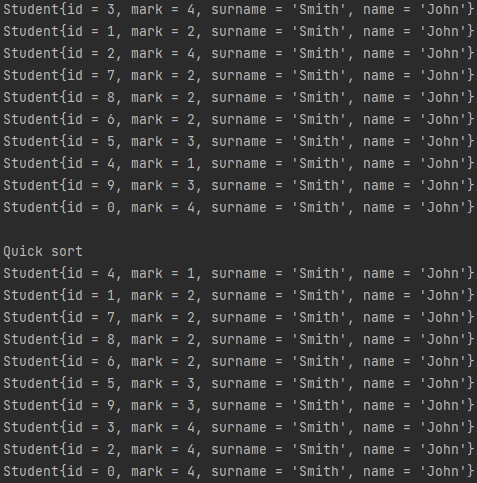


Рисунок 16 - Быстрая сортировка

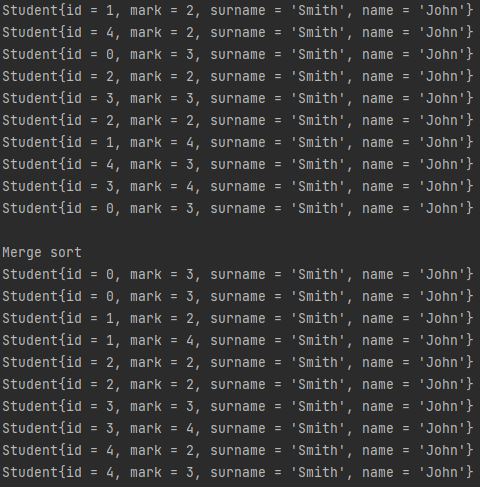


Рисунок 17 - Сортировка слиянием

## Выводы по работе:

Освоил на практике методы сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

# Практическая работа № 7

## Цель работы

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

## Теоретическое введение

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).

Vector — реализация динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Как и Hashtable, эту коллекцию не рекомендуется использовать, если не требуется достижения потокобезопасности.

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()).

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.

LinkedList — вид реализации List. Позволяет хранить любые данные, включая null. Данная коллекция реализована на основе двунаправленного связного списка

## 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».

#### 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 4.

Используйте для организации хранения структуру данных DoubleList.

### Решение:

Все представленные классы перемешивают карты игроков и записывают их в лист. Затем выбирают первую карту первого игрока и сверяют её с первой картой второго игрока, у кого карта оказалась больше, к тому и кладутся в лист обе карты. Проигрывает тот игрок, у которого оказалось 0 карт в листе, также игра заканчивается при достижении 106 хода.

#### Класс «CardGameStack»

Листинг 29 - Код "CardGameStack"

|  |
| --- |
| import java.util.Arrays;  import java.util.Collections;  import java.util.List;  import java.util.Stack;  public class CardGameStack {  public static void main(String[] args) {  int counter = 0;  //Available cards  Integer[] cards = new Integer[10];  for (int i = 0; i < 10; i++) {  cards[i] = i;  }  List<Integer> intList = Arrays.asList(cards);  Collections.shuffle(intList);  intList.toArray(cards);  Stack<Integer> firstPlayer = new Stack<>();  Stack<Integer> secondPlayer = new Stack<>();  for (int i = 0; i < 5; i++) {  firstPlayer.push(cards[i]);  secondPlayer.push(cards[5 + i]);  }  System.out.println(firstPlayer);  System.out.println(secondPlayer);  while (counter < 106 && !firstPlayer.isEmpty() && !secondPlayer.isEmpty()) {  if (firstPlayer.elementAt(0) < secondPlayer.elementAt(0)) {  secondPlayer.push(firstPlayer.elementAt(0));  secondPlayer.push(secondPlayer.elementAt(0));  } else {  firstPlayer.push(firstPlayer.elementAt(0));  firstPlayer.push(secondPlayer.elementAt(0));  }  firstPlayer.remove(0);  secondPlayer.remove(0);  counter++;  }  if (counter == 106) {  System.out.println("Botva");  } else {  if (firstPlayer.isEmpty()) {  System.out.println("second " + counter);  } else {  System.out.println("first " + counter);  }  }  }  } |

#### Класс «CardGameQueue»

Листинг 30 - Код "CardGameQueue"

|  |
| --- |
| import java.util.\*;  public class CardGameQueue {  public static void main(String[] args) {  int counter = 0;  //Available cards  Integer[] cards = new Integer[10];  for (int i = 0; i < 10; i++) {  cards[i] = i;  }  List<Integer> intList = Arrays.asList(cards);  Collections.shuffle(intList);  intList.toArray(cards);  Queue<Integer> firstPlayer = new ArrayDeque<>();  Queue<Integer> secondPlayer = new ArrayDeque<>();  for (int i = 0; i < 5; i++) {  firstPlayer.add(cards[i]);  secondPlayer.add(cards[i + 5]);  }  System.out.println(firstPlayer);  System.out.println(secondPlayer);  while (counter < 106 && !firstPlayer.isEmpty() && !secondPlayer.isEmpty()) {  if (firstPlayer.peek() < secondPlayer.peek()) {  secondPlayer.add(firstPlayer.peek());  secondPlayer.add(secondPlayer.peek());  } else {  firstPlayer.add(firstPlayer.peek());  firstPlayer.add(secondPlayer.peek());  }  firstPlayer.remove();  secondPlayer.remove();  counter++;  }  if (counter == 106) {  System.out.println("Botva");  } else {  if (firstPlayer.isEmpty()) {  System.out.println("second " + counter);  } else {  System.out.println("first " + counter);  }  }  }  } |

#### Класс «CardGameDeque»

Листинг 31 - Код "CardGameDeque"

|  |
| --- |
| import java.util.\*;  public class CardGameDeque{  public static void main(String[] args) {  int counter = 0;  //Available cards  Integer[] cards = new Integer[10];  for (int i = 0; i < 10; i++) {  cards[i] = i;  }  List<Integer> intList = Arrays.asList(cards);  Collections.shuffle(intList);  intList.toArray(cards);  Deque<Integer> firstPlayer = new LinkedList<>();  Deque<Integer> secondPlayer = new LinkedList<>();  for (int i = 0; i < 5; i++) {  firstPlayer.add(cards[i]);  secondPlayer.add(cards[i + 5]);  }  System.out.println(firstPlayer);  System.out.println(secondPlayer);  while (counter < 106 && !firstPlayer.isEmpty() && !secondPlayer.isEmpty()) {  if (firstPlayer.peek() < secondPlayer.peek()) {  secondPlayer.add(firstPlayer.peek());  secondPlayer.add(secondPlayer.peek());  } else {  firstPlayer.add(firstPlayer.peek());  firstPlayer.add(secondPlayer.peek());  }  firstPlayer.remove();  secondPlayer.remove();  counter++;  }  if (counter == 106) {  System.out.println("Botva");  } else {  if (firstPlayer.isEmpty()) {  System.out.println("second " + counter);  } else {  System.out.println("first " + counter);  }  }  }  } |

#### Класс «CardGameDoubleList»

Листинг 32 - Код "CardGameDoubleList"

|  |
| --- |
| import java.util.\*;  public class CardGameDoubleList{  public static void main(String[] args) {  int counter = 0;  //Available cards  Integer[] cards = new Integer[10];  for (int i = 0; i < 10; i++) {  cards[i] = i;  }  List<Integer> intList = Arrays.asList(cards);  Collections.shuffle(intList);  intList.toArray(cards);  LinkedList<Integer> firstPlayer = new LinkedList<>();  LinkedList<Integer> secondPlayer = new LinkedList<>();  for (int i = 0; i < 5; i++) {  firstPlayer.add(cards[i]);  secondPlayer.add(cards[i + 5]);  }  System.out.println(firstPlayer);  System.out.println(secondPlayer);  while (counter < 106 && !firstPlayer.isEmpty() && !secondPlayer.isEmpty()) {  if (firstPlayer.get(0) < secondPlayer.get(0)) {  secondPlayer.add(firstPlayer.get(0));  secondPlayer.add(secondPlayer.get(0));  } else {  firstPlayer.add(firstPlayer.get(0));  firstPlayer.add(secondPlayer.get(0));  }  firstPlayer.remove(0);  secondPlayer.remove(0);  counter++;  }  if (counter == 106) {  System.out.println("Botva");  } else {  if (firstPlayer.isEmpty()) {  System.out.println("second " + counter);  } else {  System.out.println("first " + counter);  }  }  }  } |

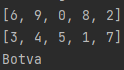


Рисунок 18 - Результат 1

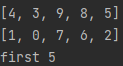


Рисунок 19 - Результат 2

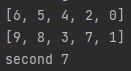


Рисунок 20 - Результат 3

## Выводы по работе:

Изучил на практике приемы работы со стандартными контейнерными классами Java Collection Framework.

# Практическая работа № 8

## Цель работы

Изучить создание и использование Generics в языке Java.

## Теоретическое введение

BoundedWaitList: Этот список ожидания имеет ограниченную емкость, указываемую в момент создания. Он не принимает более элементов, чем заранее задано (возможное количество потенциальных элементов в списке ожидания).

UnfairWaitList: В этом списке ожидания, можно удалить элемент, который не является первым в очереди - и помните он не может вернуться обратно! (Возможны различные реализации, но в вашей реализации необходимо удалить первое вхождение данного элемента.) Также возможно, чтобы, например, первый элемент будет отправлен обратно в конец списка.

После описания всей задачи в целом, мы сможем решить, что мы нам нужен интерфейс IWaitList, и затем нужно создать три разных класса для трех списков ожидания. Также предполагается, что один из списков ожидания должен быть суперклассом для двух других списков ожидания.

## Выполнение лабораторной работы

### Задание:

1. Исследуйте UML диаграмму классов на рисунке 1 и понаблюдайте, как она выражает то, что мы говорили выше в словах. Убедитесь, что вы понимаете все аспекты диаграммы.

2. Расширить и модифицировать исходный код WaitList, как необходимо, чтобы полностью реализовать всю схему UML. Включить комментарии Javadoc. Обратите внимание на переключение ролей после реализации каждого интерфейса / класса!

3. Изучение работу метода main(), которая использует ваши новые классы и интерфейс.

### Решение:

#### Интерфейс «IWaitList»

Листинг 33 - Код " IWaitList"

Интерфейс обязывает использовать методы, прописанные в нём

|  |
| --- |
| import java.util.Collection;  public interface IWaitList<E> {  void add(E element);  E remove();  boolean contains(E element);  boolean containsAll(Collection<E> c);  boolean isEmpty();  } |

#### Класс «WaitList»

Листинг 34 - Код "WaitList"

Класс создаёт лист с функциями добавления и удаления элементов, а также имеется возможность проверить, есть ли определённый элемент/другой лист в листе и не постой ли текущий лист

|  |
| --- |
| import java.util.Collection;  import java.util.concurrent.ConcurrentLinkedQueue;  /\*\*  \* Лист ожидания  \*  \* @param <E>  \*/  public class WaitList<E> implements IWaitList<E> {  protected ConcurrentLinkedQueue<E> content;  public WaitList() {  content = new ConcurrentLinkedQueue<>();  }  public WaitList(Collection<E> C) {  content = new ConcurrentLinkedQueue<>(C);  }  @Override  public void add(E element) {  content.add(element);  }  @Override  public E remove() {  return content.remove();  }  @Override  public boolean contains(E element) {  return content.contains(element);  }  @Override  public boolean containsAll(Collection<E> c) {  return content.containsAll(c);  }  @Override  public boolean isEmpty() {  return content.isEmpty();  }  @Override  public String toString() {  return "WaitList: " +  "content = " + content;  }  } |

#### Класс «BoundedWaitList»

Листинг 35 - Код "BoundedWaitList"

Тот же лист, который создаётся классом «WaitList», но ограничен в размерах

|  |
| --- |
| import java.util.Collection;  /\*\*  \* Ограниченный лист ожидания  \* @param <E>  \*/  public class BoundedWaitList<E> extends WaitList<E> {  private final int capacity;  public BoundedWaitList(int capacity) {  this.capacity = capacity;  }  public int getCapacity() {  return capacity;  }  public void add(E element) {  if (content.size() < capacity)  super.add(element);  }  @Override  public String toString() {  return "BoundedWaitList: " +  "capacity = " + capacity +  ", content = " + content;  }  } |

#### Класс «UnfairWaitList»

Листинг 36 - Код "UnfairWaitList"

Удалённый элемент отправляется в конец списка

|  |
| --- |
| /\*\*  \* Лист ожидания с возможностью удалить ожидающего  \* @param <E>  \*/  public class UnfairWaitList<E> extends WaitList<E> {  public UnfairWaitList() {  }  public E remove() {  return super.remove();  }  public void moveToBack() {  super.add(remove());  }  } |

#### Класс «Main»

Листинг 37 - Код "Main"

Проверка работы классов

|  |
| --- |
| public class Main {  public static void main(String[] args) {  WaitList<Integer> clients = new WaitList<>();  BoundedWaitList<Integer> bounded = new BoundedWaitList<>(5);  UnfairWaitList<Integer> unfair = new UnfairWaitList<>();  for (int i = 0; i < 10; i++) {  clients.add(i);  bounded.add(i);  unfair.add(i);  }  unfair.moveToBack();  System.out.println(clients);  System.out.println(bounded);  System.out.println(unfair);  }  } |

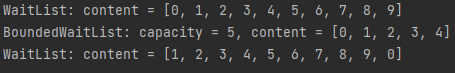


Рисунок 21 - Результат

## Выводы по работе:

Изучил создание и использование Generics в языке Java.

# Практическая работа № 9

## Цель работы

Научиться создавать собственные исключения.

## Теоретическое введение

**Создание проверяемых пользовательских исключений**

Проверяемые исключения — это исключения, которые необходимо обрабатывать явно.

Чтобы создать собственное пользовательское исключение, мы будем наследоваться от класса java.lang.Exception.

Также должны написать конструктор в нашем классе, который принимает параметр типа String в качестве сообщения об ошибке, в котором вызывается конструктор родительского класса.

**Создание непроверяемых пользовательских исключений**

Чтобы создать собственное непроверяемое исключение, нам нужно наследоваться от класса java.lang.RuntimeException

## Выполнение лабораторной работы

### Задание:

Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

### Решение:

#### Программа

Выводит ошибку, если введён неправильный ИНН

Листинг 38 - Код программы

|  |
| --- |
| import java.util.Scanner;  public class Main {  public static void main(String[] args) throws Exception {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter FIO: ");  String FIO = scanner.nextLine();  System.out.print("Enter INN: ");  String INN = scanner.nextLine();  if (!INN.matches("[0-9]{12}")) throw new Exception("Wrong INN");  }  } |

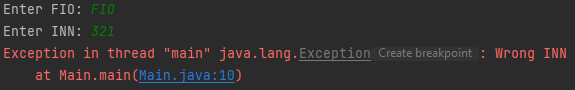


Рисунок 22 - Результат

## Выводы по работе:

Научился создавать собственные исключения.

# Практическая работа № 10

## Цель работы

Закрепить знания в области обработки строк, научиться применять методы класса String и других классов для обработки строк

## Теоретическое введение

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

Отличия класса строк от обычных классов:

• строка в Java представляет из себя строку литералов (текст), помещенных в двойные кавычки, например: "Hello, World! ". Вы можете присвоить последовательность строковых литералов непосредственно переменной типа String, вместо того чтобы вызывать конструктор для создания экземпляра класса String.

• Оператор '+' является перегруженным, для объектов типа String, и всегда используется, чтобы объединить две строки операндов. В данном контексте мы говорим об операции конкатенации или сложения строк.

• Строка является неизменяемой, то есть, символьной константой. Это значит, что ее содержание не может быть изменено после ее (строки как объекта) создания.

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

**Классы StringBuffer и StringBuilder**

Классы StringBuffer и StringBuilder в Java используются, когда возникает необходимость сделать много изменений в строке символов.

## Выполнение лабораторной работы

### Задание:

Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество.

Доработать класс адреса, который из полученной строки формата "Страна[d] Регион[d] Город[d] Улица[d] Дом[d] Корпус[d] Квартира" ([d] – разделитель, например, «запятая») выбирает соответствующие части и записывает их в соответствующие поля класса Address.

Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно.

### Решение:

#### Класс «Person»

Строит ФИО в зависимости от данных с помощью StringBuilder

Листинг 39 - Код класса Person

|  |
| --- |
| public class Person {  private String name = "", surname = "", patronymic = "";  public void setName(String name) {  this.name = name;  }  public void setSurname(String surname) {  this.surname = surname;  }  public void setPatronymic(String patronymic) {  this.patronymic = patronymic;  }  public Person() {  }  public String getFIO() {  StringBuilder stringBuilder = new StringBuilder();  stringBuilder.append(surname).append(" ");  if (!name.isEmpty()) stringBuilder.append(name.charAt(0)).append(". ");  if (!patronymic.isEmpty()) stringBuilder.append(patronymic.charAt(0)).append(".");  return stringBuilder.toString().trim();  }  } |

#### Класс «PersonTest»

Листинг 40 - Код класса PersonTest

|  |
| --- |
| public class PersonTest {  public static void main(String[] args) {  Person person = new Person();  person.setName("Nikita");  person.setPatronymic("Sergeevich");  person.setSurname("Semyannikov");  System.out.println(person.getFIO());  person.setName("");  System.out.println(person.getFIO());  person.setPatronymic("");  System.out.println(person.getFIO());  }  } |

#### Класс «Address»

Разбивает переданную строку на строки с помощью StringTokenizer

Листинг 41 - Код класса Address

|  |
| --- |
| import java.util.StringTokenizer;  public class Address {  private final String country, region, city, street, house, frame, flat;  public Address(String address) {  address = address.trim();  StringTokenizer strings = new StringTokenizer(address, " ,:.", false);  country = strings.nextToken();  region = strings.nextToken();  city = strings.nextToken();  street = strings.nextToken();  house = strings.nextToken();  frame = strings.nextToken();  flat = strings.nextToken();  }  @Override  public String toString() {  return "Address{" +  "country='" + country + '\'' +  ", region='" + region + '\'' +  ", city='" + city + '\'' +  ", street='" + street + '\'' +  ", house='" + house + '\'' +  ", frame='" + frame + '\'' +  ", flat='" + flat + '\'' +  '}';  }  } |

#### Класса «AddressTest»

Листинг 42 - Код класса AddressTest

|  |
| --- |
| public class AddressTest {  public static void main(String[] args) {  Address address = new Address("a: b: c: d e: f, g.");  System.out.println(address.toString());  address = new Address("a, b, c, d, e, f, g");  System.out.println(address.toString());  address = new Address("a. b c: d, e.: f, g");  System.out.println(address.toString());  address = new Address("a b c d e f g");  System.out.println(address.toString());  }  } |

#### Класс «Shirt»

Разбивает строку на подстроки с помощью split

Листинг 43 - Код класса Shirt

|  |
| --- |
| public class Shirt {  private String id, type, color, size;  public Shirt(String shirt) {  String[] strings = new String[4];  strings = shirt.split(",");  id = strings[0];  type = strings[1];  color = strings[2];  size = strings[3];  }  @Override  public String toString() {  return "Shirt{" +  "id='" + id + '\'' +  ", type='" + type + '\'' +  ", color='" + color + '\'' +  ", size='" + size + '\'' +  '}';  }  } |

#### Класс «ShirtTest»

Листинг 44 - Код класса ShirtTest

|  |
| --- |
| public class ShirtTest {  public static void main(String[] args) {  String[] strings = new String[11];  strings[0] = "S001,Black Polo Shirt,Black,XL";  strings[1] = "S002,Black Polo Shirt,Black,L";  strings[2] = "S003,Blue Polo Shirt,Blue,XL";  strings[3] = "S004,Blue Polo Shirt,Blue,M";  strings[4] = "S005,Tan Polo Shirt,Tan,XL";  strings[5] = "S006,Black T-Shirt,Black,XL";  strings[6] = "S007,White T-Shirt,White,XL";  strings[7] = "S008,White T-Shirt,White,L";  strings[8] = "S009,Green T-Shirt,Green,S";  strings[9] = "S010,Orange T-Shirt,Orange,S";  strings[10] = "S011,Maroon Polo Shirt,Maroon,S";  Shirt[] shirts = new Shirt[11];  for (int i = 0; i < strings.length; i++) {  shirts[i] = new Shirt(strings[i]);  System.out.println(shirts[i].toString());  }  }  } |

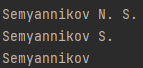


Рисунок 23 – Результат Person

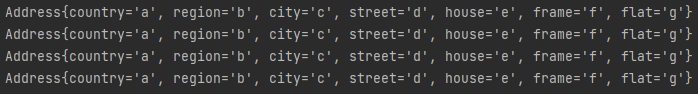


Рисунок 24 - Результат Address

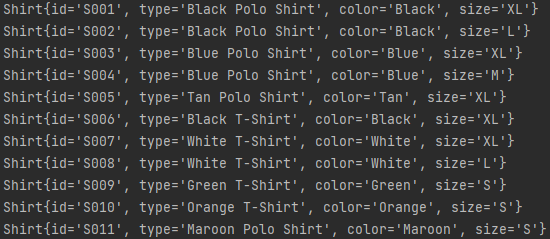


Рисунок 25 - Результат Shirt

## Выводы по работе:

Закрепил знания в области обработки строк, научился применять методы класса String и других классов для обработки строк.

# Практическая работа № 11

## Цель работы

Улучшить знания по работе с очередями и классами.

## Теоретическое введение

Инкапсуляция – размещение в одном компоненте данных и методов, которые с ними работают, а также сокрытие, позволяющее разграничивать доступ к различным компонентам программы.

Очередь – абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл – первый вышел» (FIFO, англ. first in, first out). Добавление элемента (принято обозначать словом enqueue – поставить в очередь) возможно лишь в конец очереди, выборка – только из начала очереди (что принято называть словом dequeue – убрать из очереди), при этом выбранный элемент из очереди удаляется.

Интерфейс в языке программирования Java — это абстрактный тип, используемый для описания поведения, которое должны реализовать классы.

## Выполнение лабораторной работы

### Задание:

Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.

Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).

Класс ArrayQueue должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).

Должны быть реализованы следующие функции(процедуры) / методы:

▪ enqueue – добавить элемент в очередь;

▪ element – первый элемент в очереди;

▪ dequeue – удалить и вернуть первый элемент в очереди;

▪ size – текущий размер очереди;

▪ isEmpty – является ли очередь пустой;

▪ clear – удалить все элементы из очереди.

Определите интерфейс очереди Queue и опишите его контракт.

Реализуйте класс LinkedQueue — очередь на связном списке.

Выделите общие части классов LinkedQueue и ArrayQueue в базовый класс

### Решение:

#### Класс «ArrayQueueModule»

Данный класс реализован как один экзмпляр

Листинг 45 - Код класса ArrayQueueModule

|  |
| --- |
| package Task1;  import java.util.Arrays;  public class ArrayQueueModule {  private static int size;  private static int[] elements = new int[5];  public static void enqueue(int element) {  ensureCapacity(size + 1);  elements[size++] = element;  }  private static void ensureCapacity(int capacity) {  if (capacity > elements.length) {  elements = Arrays.copyOf(elements, 2 \* capacity);  }  }  public static int element() {  assert size > 0;  return elements[0];  }  public static int dequeue() {  assert size > 0;  int first = elements[0];  for (int i = 1; i < size; i++)  elements[i - 1] = elements[i];  size--;  return first;  }  public static int size() {  return size;  }  public static boolean isEmpty() {  return size == 0;  }  public static void clear() {  elements = new int[0];  size = 0;  }  public static String toStringStat() {  StringBuilder stringBuffer = new StringBuilder();  stringBuffer.append("[ ");  for (int i = 0; i < size; i++) {  stringBuffer.append(elements[i]).append(" ");  }  stringBuffer.append("]");  return stringBuffer.toString();  }  } |

#### Класс «ArrayQueueModuleTest»

Листинг 46 - Код класса ArrayQueueModuleTest

|  |
| --- |
| package Task1;  public class ArrayQueueModuleTest {  public static void fill() {  for (int i = 0; i < 10; i++) {  ArrayQueueModule.enqueue(i);  }  }  public static void dump() {  while (!ArrayQueueModule.isEmpty()) {  System.out.println(ArrayQueueModule.toStringStat());  System.out.println("size: " + ArrayQueueModule.size() + "\n" +  "element[0] = " + ArrayQueueModule.element() + "\n" +  "delete element[0]: " + ArrayQueueModule.dequeue() + "\n"  );  }  }  public static void main(String[] args) {  fill();  dump();  }  } |

#### Класс «ArrayQueueADT»

Класс реализован в виде абстрактного типа данных, т.е. в него необходимо передавать экземпляр класса

Листинг 47 - Код класса ArrayQueueADT

|  |
| --- |
| package Task1;  import java.util.Arrays;  public class ArrayQueueADT {  private int size;  private int[] elements = new int[0];  public static void enqueue(ArrayQueueADT queue, int element) {  ensureCapacity(queue, queue.size + 1);  queue.elements[queue.size++] = element;  }  private static void ensureCapacity(ArrayQueueADT queue, int capacity) {  if (capacity > queue.elements.length) {  queue.elements = Arrays.copyOf(queue.elements, 2 \* capacity);  }  }  public static int element(ArrayQueueADT queue) {  return queue.elements[0];  }  public static int dequeue(ArrayQueueADT queue) {  int first = queue.elements[0];  for (int i = 1; i < queue.size; i++)  queue.elements[i - 1] = queue.elements[i];  queue.size--;  return first;  }  public static int size(ArrayQueueADT queue) {  return queue.size;  }  public static boolean isEmpty(ArrayQueueADT queue) {  return queue.size == 0;  }  public static void clear(ArrayQueueADT queue) {  queue.elements = new int[0];  queue.size = 0;  }  public static String toStringStat(ArrayQueueADT queue) {  StringBuilder stringBuffer = new StringBuilder();  stringBuffer.append("[ ");  for (int i = 0; i < queue.size; i++) {  stringBuffer.append(queue.elements[i]).append(" ");  }  stringBuffer.append("]");  return stringBuffer.toString();  }  } |

#### Класс «ArrayQueueADTTest»

Листинг 48 - ArrayQueueADTTest

|  |
| --- |
| package Task1;  import java.util.Random;  public class ArrayQueueADTTest {  public static void fill(ArrayQueueADT queueADT) {  Random rand = new Random();  for (int i = 0; i < 10; i++) {  queueADT.enqueue(queueADT, rand.nextInt(100));  }  }  public static void dump(ArrayQueueADT queueADT) {  while (!ArrayQueueADT.isEmpty(queueADT)) {  System.out.println(ArrayQueueADT.toStringStat(queueADT));  System.out.println(  "size: " + ArrayQueueADT.size(queueADT) + "\n"  + "element[0] = " + ArrayQueueADT.element(queueADT) + "\n"  + "delete element[0]: " + ArrayQueueADT.dequeue(queueADT) + "\n"  );  }  }  public static void main(String[] args) {  ArrayQueueADT queueADT = new ArrayQueueADT();  fill(queueADT);  dump(queueADT);  }  } |

#### Класс «ArrayQueueModuleTest»

Класс реализован в виде класса, т.е. с неявной передачей данных

Листинг 49 - Код класса ArrayQueueModuleTest

|  |
| --- |
| package Task1;  import java.util.Arrays;  public class ArrayQueue {  private int size;  private int[] elements = new int[0];  public void enqueue(int element) {  ensureCapacity(size + 1);  elements[size++] = element;  }  private void ensureCapacity(int capacity) {  if (capacity > elements.length) {  elements = Arrays.copyOf(elements, 2 \* capacity);  }  }  public int element() {  assert size > 0;  return elements[0];  }  public int dequeue() {  assert size > 0;  int first = elements[0];  for (int i = 1; i < size; i++)  elements[i - 1] = elements[i];  size--;  return first;  }  public int size() {  return size;  }  public boolean isEmpty() {  return size == 0;  }  public void clear() {  elements = new int[0];  size = 0;  }  public String toString() {  StringBuffer stringBuffer = new StringBuffer();  stringBuffer.append("[ ");  for (int i = 0; i < size; i++) {  stringBuffer.append(elements[i] + " ");  }  stringBuffer.append("]");  return stringBuffer.toString();  }  } |

#### Класс «ArrayQueueTest»

Листинг 50 - Код класса ArrayQueueTest

|  |
| --- |
| package Task1;  import java.util.Random;  public class ArrayQueueTest {  public static void fill(ArrayQueue queue) {  Random rand = new Random();  for (int i = 0; i < 10; i++) {  queue.enqueue(rand.nextInt(100));  }  }  public static void dump(ArrayQueue queue) {  while (!queue.isEmpty()) {  System.out.println(queue.toString());  System.out.println(  "size: " + queue.size() + "\n"  + "element[0] = " + queue.element() + "\n"  + "delete element[0]: " + queue.dequeue() + "\n"  );  }  }  public static void main(String[] args) {  ArrayQueue queue = new ArrayQueue();  fill(queue);  dump(queue);  }  } |

#### Интерфейс «Queue»

Интерфейс, включающий в себя основные методы очереди на основе листа

Листинг 51 - Код интерфейса Queue

|  |
| --- |
| package Task2;  public interface Queue<E> {  E element();  boolean push(E elem);  E pop();  int size();  boolean isEmpty();  } |

#### Класс «AbstractQueue»

Класс включает в себя общие методы классов ArrayQueue и LinkedQueue

Листинг 52 - Код класса AbstractQueue

|  |
| --- |
| package Task2;  public abstract class AbstractQueue<E> implements Queue<E> {  protected int size;  public E element() {  assert size > 0;  return getElement();  }  public boolean push(E element) {  assert element != null;  pushElem(element);  size++;  return true;  }  public E pop() {  assert size > 0;  E result = element();  size--;  remove();  return result;  }  public int size() {  return size;  }  public boolean isEmpty() {  return size == 0;  }  protected abstract void pushElem(E element);  protected abstract void remove();  protected abstract E getElement();  } |

#### Класс «ArrayQueue»

Класс представляет очередь

Листинг 53 - Код класса ArrayQueue

|  |
| --- |
| package Task2;  import java.util.ArrayList;  public class ArrayQueue<E> extends AbstractQueue<E> {  private final ArrayList<E> elements = new ArrayList<>();  @Override  protected void pushElem(E element) {  elements.add(element);  }  @Override  protected void remove() {  elements.remove(0);  }  @Override  protected E getElement() {  return elements.get(0);  }  } |

#### Класс «LinkedQueue»

Класс представляет очередь на основе листа

Листинг 54 - Код класса LinkedQueue

|  |
| --- |
| package Task2;  public class LinkedQueue<E> extends AbstractQueue<E> {  private class Node {  private final E value;  private Node next;  public Node(E value, Node next) {  assert value != null;  this.value = value;  this.next = next;  }  }  private Node start;  private Node end;  protected void pushElem(E element) {  if (start == null) {  start = new Node(element, null);  end = start;  } else {  Node node = new Node(element, null);  end.next = node;  end = node;  }  }  @Override  protected void remove() {  if (size == 0) {  start = null;  end = null;  } else {  start = start.next;  }  }  @Override  protected E getElement() {  return start.value;  }  } |

#### Класс «QueueTests»

Листинг 55 - Код класса QueueTests

|  |
| --- |
| package Task2;  public class QueueTests {  public static void fill(Queue queue) {  for (int i = 0; i < 3; i++) {  queue.push(i);  }  }  public static void dump(Queue queue) {  while (!queue.isEmpty()) {  System.out.println("Size: " + queue.size() + "; First: " +  queue.element() + "; Removing: " + queue.pop());  }  }  public static void test(Queue queue) {  fill(queue);  dump(queue);  System.out.println();  }  public static void main(String[] args) {  System.out.println("Array");  test(new ArrayQueue<Integer>());  System.out.println("Linked");  test(new LinkedQueue<Integer>());  }  } |

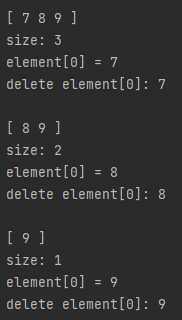


Рисунок 26 - Результат ArrayQueueModuleTest

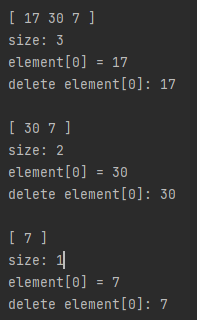


Рисунок 27 - Результат ArrayQueueADTTest

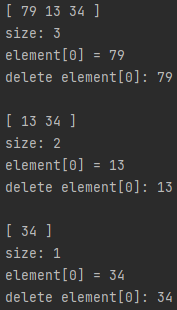


Рисунок 28 - Результат ArrayQueueTest

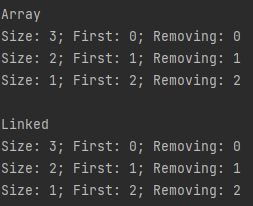


Рисунок 29 - Результат QueueTests

## Выводы по работе:

Укрепил знания в работе классов, листов и очередей.

# Практическая работа № 12

## Цель работы

Закрепить знания в области обработки строк, научиться применять методы класса String и других классов для обработки строк

## Теоретическое введение

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

Отличия класса строк от обычных классов:

• строка в Java представляет из себя строку литералов (текст), помещенных в двойные кавычки, например: "Hello, World! ". Вы можете присвоить последовательность строковых литералов непосредственно переменной типа String, вместо того чтобы вызывать конструктор для создания экземпляра класса String.

• Оператор '+' является перегруженным, для объектов типа String, и всегда используется, чтобы объединить две строки операндов. В данном контексте мы говорим об операции конкатенации или сложения строк.

• Строка является неизменяемой, то есть, символьной константой. Это значит, что ее содержание не может быть изменено после ее (строки как объекта) создания.

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

**Классы StringBuffer и StringBuilder**

Классы StringBuffer и StringBuilder в Java используются, когда возникает необходимость сделать много изменений в строке символов.

## Выполнение лабораторной работы

### Задание:

Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов:

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

“+104289652211” или

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

и преобразует полученную строку в формат:

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

В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

### Решение:

#### Класс «Phone»

Класс собирает номер телефона используя переданную строку с помощью StringBuilder

Листинг 56 - Код класса Phone

|  |
| --- |
| public class Phone {  String phone;  public String getPhone() {  return phone;  }  public Phone(String phone) {  int len = phone.length();  phone = new StringBuilder(phone).insert(phone.length() - 4, "-").insert(phone.length() - 7, "-").toString();  if (phone.charAt(0) != '+') {  phone = phone.replace(phone.charAt(0), (char) ((int) phone.charAt(0) - 1));  phone = new StringBuilder(phone).insert(0, "+").toString();  }  this.phone = phone;  }  } |

#### Класс «PhoneTest»

Листинг 57 - Код класса PhoneTest

|  |
| --- |
| public class PhoneTest {  public static void main(String[] args) {  Phone phoneNumber = new Phone("+79175655655");  System.out.println(phoneNumber.getPhone());  phoneNumber = new Phone("+104289652211");  System.out.println(phoneNumber.getPhone());  phoneNumber = new Phone("89175655655");  System.out.println(phoneNumber.getPhone());  }  } |

#### Класс «Main»

Класс принимает название файла, записывает данные файла в строку, а затем разбивает строку на массив слов. Массив слов прогоняется по циклу, проверяя последний символ текущего слово и первый символ слова, выбранного циклом. При совпадении символов найденное слово добавляется в строку, а текущее удаляется.

Листинг 58 - Код класса Main

|  |
| --- |
| import java.io.FileNotFoundException;  import java.io.FileReader;  import java.util.ArrayList;  import java.util.Scanner;  import java.util.StringTokenizer;  public class Main {  public static FileReader reader;  private static String getWords() {  Scanner scanner = new Scanner(reader);  StringBuilder string = new StringBuilder().append(scanner.nextLine());  StringTokenizer stringTokenizer = new StringTokenizer(string.toString(), " ", false);  ArrayList<String> words = new ArrayList<String>();  int wordsCount = stringTokenizer.countTokens();  for (int i = 0; i < wordsCount; i++) {  words.add(stringTokenizer.nextToken());  }  int step = 0;  string = new StringBuilder();  String last = "";  if (words.size() > 0) {  string.append(words.get(0));  last = words.get(0);  words.remove(0);  }  while (step != wordsCount && words.size() != 0) {  for (int i = 0; i < words.size(); i++) {  if (last.toLowerCase().charAt(last.length() - 1) == words.get(i).toLowerCase().charAt(0)) {  string.append(" ").append(words.get(i));  last = words.get(i);  words.remove(i);  break;  }  }  step += 1;  }  return string.toString();  }  public static void main(String[] args) throws FileNotFoundException {  Scanner scanner = new Scanner(System.in);  System.out.print("File name: ");  String filename = scanner.nextLine();  reader = new FileReader(filename);  System.out.println(getWords());  }  } |

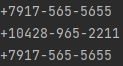


Рисунок 30 - Результат PhoneTest



Рисунок 31 - Результат Main



Рисунок 32 - Текст в файле

## Выводы по работе:

Закрепил знания в области обработки строк, научился применять методы класса String и других классов для обработки строк.

# Практическая работа № 13

## Цель работы

Изучить фабрики

## Теоретическое введение

Фабрика – это объект для создания других объектов. Формально фабрика – это функция или метод, который возвращает объекты изменяющегося прототипа или класса из некоторого вызова метода, который считается «новым».

## Выполнение лабораторной работы

### Задание:

Реализовать класс Абстрактная фабрика для комплексных чисел

### Решение:

#### Класс «Complex»

Класс содержит значения комплексного числа

Листинг 59 - Код класса Complex

|  |
| --- |
| public class Complex {  private int real, image;  public Complex() {  }  public Complex(int real, int image) {  this.real = real;  this.image = image;  }  public String toString() {  return real + String.format(" + %.0f", (float) image).toString() + "i";  }  } |

#### Интерфейс «ComplexAbstractFactory»

Содержит методы для класса ConcreteFactory

Листинг 60 - Код класса ComplexAbstractFactory

|  |
| --- |
| public interface ComplexAbstractFactory {  public Complex createComplex();  public Complex createComplex(int real, int image);  } |

#### Класс «ConcreteFactory»

Создаёт элементы типа Complex

Листинг 61 - Код класса ConcreteFactory

|  |
| --- |
| public class ConcreteFactory implements ComplexAbstractFactory {  @Override  public Complex createComplex() {  return new Complex();  }  @Override  public Complex createComplex(int real, int image) {  return new Complex(real, image);  }  } |

#### Класс «Main»

Листинг 62 - Код класса Main

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Complex complex;  ConcreteFactory concreteFactory = new ConcreteFactory();  complex = concreteFactory.createComplex(1, 2);  System.out.println(complex.toString());  complex = concreteFactory.createComplex();  System.out.println(complex.toString());  }  } |



Рисунок 33 - Результат

## Выводы по работе:

Изучил фабрики

# Практическая работа № 14

## Цель работы

Укрепить знания о фабриках

## Теоретическое введение

Абстрактная фабрика — порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы.

## Выполнение лабораторной работы

### Задание:

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

### Решение:

#### Интерфейс «AbstractChairFactory»

Содержит методы для дальнейших классов

Листинг 63 - Код класса AbstractChairFactory

|  |
| --- |
| public interface AbstractChairFactory {  public VictorianChair createVictorianChair();  public MagicChair createMagicChair();  public FunctionalChair createFunctionalChair();  } |

#### Класс «Chair»

Листинг 64 - Код класса Chair

|  |
| --- |
| public interface Chair {  } |

#### Класс «ChairFactory»

Фабрика для всех стульев

Листинг 65 - Код класса ChairFactory

|  |
| --- |
| public class ChairFactory implements AbstractChairFactory{  @Override  public VictorianChair createVictorianChair() {  VictorianChair victorianChair;  return new VictorianChair();  }  @Override  public MagicChair createMagicChair() {  return new MagicChair();  }  @Override  public FunctionalChair createFunctionalChair() {  return new FunctionalChair();  }  } |

#### Класс «FunctionalChair»

Стул, который умеет суммировать числа

Листинг 66 - Код класса FunctionalChair

|  |
| --- |
| public class FunctionalChair implements Chair{  public int sum(int a, int b) {  return a + b;  }  } |

#### Класс «MagicChair»

Стул, знающий магию

Листинг 67 - Код класса MagicChair

|  |
| --- |
| public class MagicChair implements Chair{  public void doMagic() {  System.out.println("(∩｀-´)⊃━☆ﾟ.\*･｡ﾟ");  }  } |

#### Класс «VictorianChair»

Древний стул

Листинг 68 - Код класса VictorianChair

|  |
| --- |
| public class VictorianChair implements Chair {  private int age;  public VictorianChair(int age) {  this.age = age;  }  public VictorianChair() {  }  public int getAge() {  return age;  }  } |

#### Класс «Client»

Клиент, который может садиться на стулья

Листинг 69 - Код класса Client

|  |
| --- |
| public class Client {  public Chair chair;  public void sit() {  System.out.println("Chairs... They are everywhere...");  }  public void setChair(Chair chair) {  this.chair = chair;  }  } |

#### Класс «Main»

Листинг 70 - Код класса Main

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Client client = new Client();  ChairFactory chairFactory = new ChairFactory();  client.sit();  client.setChair(chairFactory.createMagicChair());  ((MagicChair) client.chair).doMagic();  client.setChair(chairFactory.createFunctionalChair());  System.out.println(((FunctionalChair) client.chair).sum(1, 2));  client.setChair(chairFactory.createVictorianChair());  }  } |

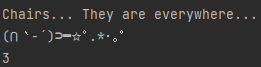


Рисунок 34 - Результат

## Выводы по работе:

Укрепил знания о фабриках

# Практическая работа № 15

## Цель работы

Реализовать работу с документами с использованием фабрик.

## Теоретическое введение

Фабрика – это объект для создания других объектов. Формально фабрика – это функция или метод, который возвращает объекты изменяющегося прототипа или класса из некоторого вызова метода, который считается «новым».

Абстрактная фабрика — порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы.

## Выполнение лабораторной работы

### Задание:

Создать каркас приложения — окно редактора с меню File. В меню File реализовать пункты New, Open, Save, Exit.

Продемонстрировать работу каркаса на примере текстового редактора. Потребуется создать конкретный унаследованный класс TextDocument и фабрику для него — CreateTextDocument.

### Решение:

Интерфейс «IDocument»

Хранит создание формы и основное методы для создания и открытия файлов

Листинг 71 - Код класса IDocument

|  |
| --- |
| import javax.swing.\*;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  public interface IDocument {  public default JFrame document() {  JFrame jFrame = new JFrame();  JMenuBar jMenuBar = new JMenuBar();  JMenu menuFile = new JMenu("File");  JMenu menuNew = new JMenu("New");  JMenu menuOpen = new JMenu("Open");  JMenuItem menuSave = new JMenuItem("Save");  JMenuItem menuExit = new JMenuItem("Exit");  JMenuItem music = new JMenuItem("mp3");  JMenuItem image = new JMenuItem("PNG");  JMenuItem text = new JMenuItem("text");  JMenuItem omusic = new JMenuItem("mp3");  JMenuItem oimage = new JMenuItem("PNG");  JMenuItem otext = new JMenuItem("text");  //  menuFile.add(menuNew);  menuFile.add(menuOpen);  menuFile.add(menuSave);  menuFile.add(menuExit);  //New  menuNew.add(music);  menuNew.add(image);  menuNew.add(text);  //Open  menuOpen.add(omusic);  menuOpen.add(oimage);  menuOpen.add(otext);  //Text document  text.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  jFrame.dispose();  ((TextDocument) new CreateTextDocument().CreateNew()).documentTXT();  }  });  otext.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  jFrame.dispose();  ((TextDocument) new CreateTextDocument().CreateNew()).documentTXT();  }  });  //Image  image.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  jFrame.dispose();  ((ImageDocument) new CreateImageDocument().CreateNew()).documentIMG();  }  });  oimage.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  jFrame.dispose();  ((ImageDocument) new CreateImageDocument().CreateNew()).documentIMG();  }  });  //Music  music.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(null, "Currently unavailable!", "Warning!", JOptionPane.INFORMATION\_MESSAGE);  }  });  omusic.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(null, "Currently unavailable!", "Warning!", JOptionPane.INFORMATION\_MESSAGE);  }  });  //Exit  menuExit.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  jFrame.dispose();  }  });  //Saving  menuSave.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(null, "Saved!", "Save", JOptionPane.INFORMATION\_MESSAGE);  }  });  jMenuBar.add(menuFile);  jFrame.setJMenuBar(jMenuBar);  jFrame.setSize(400, 400);  jFrame.setVisible(true);  jFrame.setLocationRelativeTo(null);  jFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  return jFrame;  }  } |

#### Класс «ICreateDocument»

Хранит методы для создания/открытия документов

Листинг 72 - Код класса ICreateDocument

|  |
| --- |
| public interface ICreateDocument {  public IDocument CreateNew();  public IDocument CreateOpen();  } |

#### Класс «CreateTextDocument»

Создаёт/открывает текстовый документ

Листинг 73 - Код класса CreateTextDocument

|  |
| --- |
| public class CreateTextDocument implements ICreateDocument {  @Override  public IDocument CreateNew() {  return new TextDocument();  }  @Override  public IDocument CreateOpen() {  return new TextDocument();  }  } |

#### Класс «CreateImageDocument»

Создаёт/открывает изображение

Листинг 74 - Код класса CreateImageDocument

|  |
| --- |
| public class CreateImageDocument implements ICreateDocument {  @Override  public IDocument CreateNew() {  return new ImageDocument();  }  @Override  public IDocument CreateOpen() {  return new ImageDocument();  }  } |

#### Класс «TextDocument»

Создаёт новую форму с текстовым полем

Листинг 75 - Код класса TextDocument

|  |
| --- |
| import javax.swing.\*;  public class TextDocument extends JFrame implements IDocument{  public void documentTXT() {  JFrame jFrame = (IDocument.super.document());  JTextField jTextField = new JTextField();  jFrame.add(jTextField);  jFrame.setVisible(true);  }  } |

#### Класс «ImageDocument»

Создаёт новую форму с картинкой

Листинг 76 - Код класса ImageDocument

|  |
| --- |
| import javax.swing.\*;  import java.awt.\*;  public class ImageDocument implements IDocument {  public void documentIMG() {  JFrame jFrame = (IDocument.super.document());  JLabel jLabel = new JLabel();  ImageIcon imageIcon = new ImageIcon(new ImageIcon("img.png").getImage().getScaledInstance(400, 400, Image.SCALE\_DEFAULT));  jLabel.setIcon(imageIcon);  jFrame.add(jLabel);  jFrame.setVisible(true);  }  } |

#### Класс «MusicDocument»

Класс для будущей музыки

Листинг 77 - Код класса MusicDocument

|  |
| --- |
| public class MusicDocument implements IDocument {  } |

#### Класс « Main»

Листинг 78 - Код класса Main

|  |
| --- |
| import javax.swing.\*;  public class Main {  public static void main(String[] args) {  IDocument iDocument = new IDocument() {  @Override  public JFrame document() {  IDocument.super.document();  return null;  }  };  iDocument.document();  }  } |

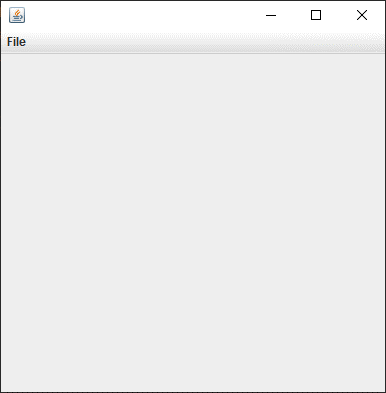


Рисунок 35 - Основное окно

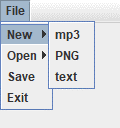


Рисунок 36 - Создание элементов

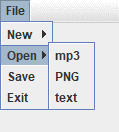


Рисунок 37 - Открытие элементов

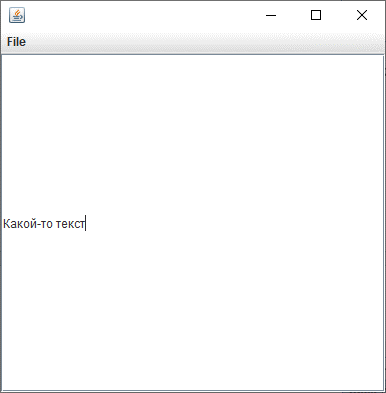


Рисунок 38 - Форма с текстовым документом

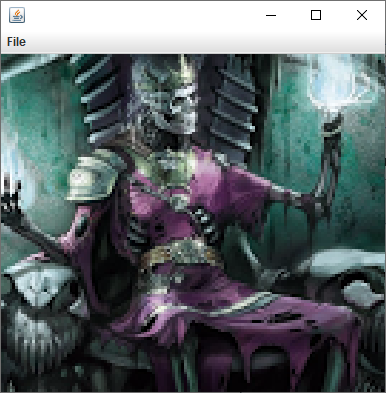


Рисунок 39 - Форма с картинкой

## Выводы по работе:

Реализовал работу с документами с использованием фабрик

# Практическая работа № 16

## Цель работы

Ознакомиться с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов

## Теоретическое введение

Класс в Java — это шаблон для создания объекта, а объект — это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java.

Интерфейс в языке программирования Java — это абстрактный тип, используемый для описания поведения, которое должны реализовать классы.

Исключение – это нежелательная ситуация, которая возникает во время выполнения программы и нарушает нормальный ход ее работы.

## Выполнение лабораторной работы

### Задание:

Создайте класс Drink – напитка. Класс описывает сущность – напиток и характеризуется следующими свойствами - стоимостью, названием и описанием.

Создайте интерфейс Item – для работы с позициями заказа. Интерфейс определяет 3 метода:

возвращает стоимость.

возвращает название.

возвращает описание позиции.

Создайте класс InternetOrder, который моделирует сущность интернет-заказ в ресторане или кафе. Класс основан на циклическом двусвязном списке с выделенной головой и может хранить как блюда, так и напитки.

### Решение:

#### Класс «Item»

Содержит методы для дальнейших классов

Листинг 79 - Код класса Item

|  |
| --- |
| public interface Item {  int getPrice();  String getTitle();  String getDescription();  } |

#### Класс «Dish»

Содержит параметры блюда

Листинг 80 - Код класса Dish

|  |
| --- |
| final class Dish implements Item {  private final String title;  private final String description;  private final int price;  public Dish(String title, String description, int price) {  this.title = title;  this.description = description;  this.price = price;  }  public Dish(String title, String description) {  this.title = title;  this.description = description;  price = 0;  }  @Override  public int getPrice() {  return price;  }  @Override  public String getTitle() {  return title;  }  @Override  public String getDescription() {  return description;  }  } |

#### Класс «Drink»

Содержит параметры напитка

Листинг 81 - Код класса Drink

|  |
| --- |
| final class Drink implements Item {  private final String title;  private final String description;  private final int price;  public Drink(String title, String description, int price) {  this.title = title;  this.description = description;  this.price = price;  }  public Drink(String title, String description) {  this.title = title;  this.description = description;  price = 0;  }  public String getTitle() {  return title;  }  public String getDescription() {  return description;  }  public int getPrice() {  return price;  }  } |

#### Класс «Order»

Содержит информацию о заказанных товарах

Листинг 82 - Код класса Order

|  |
| --- |
| public class Order {  private Item[] items;  private int capacity;  private int size = 0;  public Order(int capacity) {  this.capacity = capacity;  items = new Item[capacity];  }  public Item getItem(int index) {  try {  return items[index];  } catch (IndexOutOfBoundsException e) {  System.out.println("Out of bounds");  }  return null;  }  public void addItem(Item item) {  if (size == capacity - 1) {  capacity \*= 2;  copy(capacity);  }  items[size] = item;  size++;  }  public int size() {  return size;  }  private void copy(int newCapacity) {  Item[] newItems = new Item[newCapacity];  if (size >= 0) System.arraycopy(items, 0, newItems, 0, size);  items = newItems;  }  public void remove(Item item) {  for (int i = 0; i < size; i++)  if (items[i] == item) {  for (int j = i; j < size - 1; j++)  items[j] = items[j + 1];  size--;  }  }  } |

#### Класс «TablesOrderManager»

Содержит информацию о заказах

Листинг 83 - Код класса TablesOrderManager

|  |
| --- |
| public class TablesOrderManager {  private Order[] orders;  private int capacity;  private int size = 0;  public TablesOrderManager(int capacity) {  this.capacity = capacity;  orders = new Order[capacity];  }  public void addOrder(Order order) {  if (size == capacity - 1) {  capacity \*= 2;  copy(capacity);  }  orders[size] = order;  size += 1;  }  public Order getOrder(int index) {  try {  return orders[index];  } catch (IndexOutOfBoundsException e) {  System.out.println("Out of bounds");  }  return null;  }  public int getSize() {  return size;  }  private void copy(int newCapacity) {  Order[] newOrders = new Order[newCapacity];  if (size >= 0) System.arraycopy(orders, 0, newOrders, 0, size);  orders = newOrders;  }  public void remove(Order order) {  for (int i = 0; i < size; i++)  if (orders[i] == order) {  for (int j = i; j < size; j++)  orders[j] = orders[j - 1];  size -= 1;  }  }  } |

#### Класс «InternetOrder»

Реализует двусвязный циклический список, в котором находятся блюда и напитки

Листинг 84 - Код класса InternetOrder

|  |
| --- |
| public class InternetOrder<Item> {  InternetOrder() {  }  InternetOrder(Item[] items) {  for (Item item : items) {  add(item);  }  }  private class Node {  private Node next;  private Node prev;  private Item value;  Node(Node next, Node prev, Item value) {  this.next = next;  this.prev = prev;  this.value = value;  }  Node(Item value) {  this(null, null, value);  }  }  private Node first;  private Node last;  private int size;  //Creating node  public boolean add(Item item) {  if (first == null) {  first = new Node(item);  first.prev = first.next = first;  last = first;  } else {  Node node = new Node(first, last, item);  last.next = first.prev = node;  first = node;  }  size++;  return true;  }  private boolean isEmpty() {  return this.size == 0;  }  private Item removeFirst() {  Item oldElement;  if (this.size == 1) {  oldElement = this.first.value;  } else {  oldElement = this.first.value;  Node newFirst = this.first.next;  newFirst.prev = this.last;  this.last.next = newFirst;  this.first = newFirst;  this.size--;  }  return oldElement;  }  private Item removeLast() {  Item oldElement;  if (this.size == 1) {  oldElement = this.first.value;  } else {  oldElement = this.last.value;  Node newLast = this.last.prev;  newLast.next = this.first;  this.first.prev = newLast;  this.last = newLast;  this.size--;  }  return oldElement;  }  //Remove element by name  public boolean remove(String name) {  boolean result = false;  if (!isEmpty()) {  if (((Dish) convertToDish(this.first.value)).getTitle().equals(name)) {  removeFirst();  result = true;  } else {  Node delete = null;  for (Node node = this.first.next; node != this.first; node = node.next) {  if (((Dish) convertToDish(node.value)).getTitle().equals(name)) {  delete = node;  }  }  if (delete != null) {  if (delete == this.last) {  removeLast();  result = true;  } else {  delete.prev.next = delete.next;  delete.next.prev = delete.prev;  this.size--;  result = true;  }  }  }  }  return result;  }  //Searching element with given name  private boolean indexOf(String name) {  boolean result = false;  if (!isEmpty()) {  if (((Dish) convertToDish(this.first.value)).getTitle().equals(name)) {  result = true;  } else {  int count = 1;  for (Node node = this.first.next; node != this.first; node = node.next) {  if (((Dish) convertToDish(node.value)).getTitle().equals(name)) {  result = true;  break;  } else {  count++;  }  }  }  }  return result;  }  //Remove all elements with given name  public int removeAll(String name) {  Dish dish = new Dish(name, "");  int count = 0;  while (indexOf(name)) {  count++;  remove(name);  }  return count;  }  public int totalCount() {  int count = 0;  for (Node node = this.first.next; node != this.first; node = node.next) {  count++;  }  return count;  }  public int totalCost() {  int cost = 0;  for (Node node = this.first.next; node != this.first; node = node.next) {  cost += ((Dish) convertToDish(node.value)).getPrice();  }  return cost;  }  private Item convertToDish(Item item) {  Dish dish;  if (item.getClass().equals(Drink.class)) {  dish = new Dish(((Drink) item).getTitle(), ((Drink) item).getDescription(), ((Drink) item).getPrice());  } else {  dish = new Dish(((Dish) item).getTitle(), ((Dish) item).getDescription(), ((Dish) item).getPrice());  }  return (Item) dish;  }  //Convert list to array  public Item[] getArray() {  Item[] items = (Item[]) new Dish[size];  int i = 0;  items[i] = convertToDish(first.value);  for (Node node = this.first.next; node != this.first; node = node.next) {  i++;  items[i] = convertToDish(node.value);  }  return items;  }  //Gets amount of elements with given name  public int numberOf(String name) {  int count = 0;  if (((Dish) convertToDish(this.first.value)).getTitle().equals(name)) count++;  for (Node node = this.first.next; node != this.first; node = node.next) {  if (((Dish) convertToDish(node.value)).getTitle().equals(name)) count++;  }  return count;  }  //Get array with unique elements  public Item[] getUniqueArray() {  Item[] items = (Item[]) new Dish[size];  int i = 0;  items[i] = convertToDish(first.value);  i++;  boolean unique;  for (Node node = this.first.next; node != this.first; node = node.next) {  unique = true;  for (int j = 0; j < i; j++) {  if (((Dish) items[j]).getTitle() == ((Dish) convertToDish(node.value)).getTitle() &&  ((Dish) items[j]).getDescription() == ((Dish) convertToDish(node.value)).getDescription() &&  ((Dish) items[j]).getPrice() == ((Dish) convertToDish(node.value)).getPrice()) {  unique = false;  break;  }  }  if (unique) {  items[i] = convertToDish(node.value);  i++;  }  }  return items;  }  //Get sorted array  public Item[] getSortArray() {  Item[] items = getArray();  for (int i = 0; i < items.length - 1; i++) {  for (int j = 0; j < items.length - 1; j++) {  if (((Dish) items[j]).getPrice() > ((Dish) items[j + 1]).getPrice()) {  Item temp = items[j];  items[j] = items[j + 1];  items[j + 1] = temp;  }  }  }  return items;  }  } |

#### Класс «Main»

Листинг 85 - Код класса Main

|  |
| --- |
| public class Main {  public static void displayArray(InternetOrder<Item> internetOrder) {  Item[] items = internetOrder.getArray();  for (int i = 0; i < items.length; i++) {  System.out.println(items[i].getTitle() + " " + items[i].getDescription() + " " + items[i].getPrice());  }  System.out.println();  }  public static void main(String[] args) {  InternetOrder<Item> internetOrder = new InternetOrder<Item>();  Item item = new Dish("First", "text", 2000);  internetOrder.add(item);  item = new Dish("Second", "another text", 1500);  internetOrder.add(item);  item = new Drink("Mojito", "another text", 3000);  internetOrder.add(item);  item = new Dish("Second", "another text", 1500);  internetOrder.add(item);  internetOrder.add(item);  System.out.println("Number of: " + internetOrder.numberOf("Second"));  System.out.println("Total cost: " + internetOrder.totalCost());  System.out.println("Unique array");  Item[] items = internetOrder.getUniqueArray();  for (int i = 0; i < items.length && items[i] != null; i++) {  System.out.println(items[i].getTitle() + " " + items[i].getDescription() + " " + items[i].getPrice());  }  System.out.println("\nSorted array");  items = internetOrder.getSortArray();  for (int i = 0; i < items.length; i++) {  System.out.println(items[i].getTitle() + " " + items[i].getDescription() + " " + items[i].getPrice());  }  System.out.println("\nSimple array");  displayArray(internetOrder);  System.out.println("\nRemoving one");  internetOrder.remove("Second");  displayArray(internetOrder);  System.out.println("Removing all");  internetOrder.removeAll("Second");  displayArray(internetOrder);  }  } |

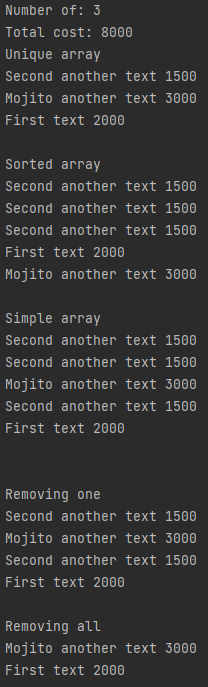


Рисунок 40 - Результат

## Выводы по работе:

Ознакомился с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов

# Практическая работа № 17

## Цель работы

Переделать практическую работу №16

## Теоретическое введение

Класс в Java — это шаблон для создания объекта, а объект — это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java.

Интерфейс в языке программирования Java — это абстрактный тип, используемый для описания поведения, которое должны реализовать классы.

Исключение – это нежелательная ситуация, которая возникает во время выполнения программы и нарушает нормальный ход ее работы.

## Выполнение лабораторной работы

### Задание:

Переименуйте класс Order из предыдущего задания в RestaurantOrder.

Создайте интерфейс Order – позиции заказа.

Интерфейс должен определять следующие методы:

добавления позиции в заказ (принимает ссылку типа Item), при этом возвращает логическое значение.

удаляет позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Возвращает логическое значение.

удаляет все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов.

возвращает общее число позиций заказа в заказе.

возвращает массив позиций заказа.

возвращает общую стоимость заказа.

возвращает число заказанных блюд или напитков (принимает название в качестве параметра).

возвращает массив названий заказанных блюд и напитков (без повторов).

возвращает массив позиций заказа, отсортированный по убыванию цены.

Переименуйте класс TablesOrderManager в OrderManager. Добавьте ему еще одно поле типа java.util.HasMap<String, Order>, которое содержит пары адрес-заказ, и методы (работающие с этим полем):

Методы класса:

перегрузка метода добавления заказа. В качестве параметров принимает строку – адрес и ссылку на заказ.

перегрузка метода получения заказа. В качестве параметра принимает строку – адрес.

перегрузка метода удаления заказа. В качестве параметра принимает строку – адрес заказа.

перегрузка метода добавления позиции к заказу. В качестве параметра принимает адрес и Item.

возвращающий массив имеющихся на данный момент интернет-заказов.

возвращающий суммарную сумму имеющихся на данный момент интернет-заказов.

возвращающий общее среди всех интернет-заказов количество заказанных порций заданного блюда по его имени

Создайте объявляемое исключение OrderAlreadyAddedException, выбрасываемое при попытке добавить заказ столику или по адресу, если со столиком или адресатом уже связан заказ

### Решение:

#### Класс «Order»

Хранит методы для дальнейших классов

Листинг 86 - Код класса Order

|  |
| --- |
| public interface Order {  boolean add(Item item);  boolean remove(String name);  int removeAll(String name);  int totalCount();  Item[] getArray();  int totalCost();  int numberOf(String name);  Item[] getUnicArray();  Item[] getSortArray();  } |

#### Класс «RestaurantOrder»

Реализует то же, что и Order из практической №16, но через интерфейс

Листинг 87 - Код класса RestaurantOrder

|  |
| --- |
| import java.util.Objects;  public class RestaurantOrder implements Order{  private Item[] items;  private int capacity;  private int size = 0;  public RestaurantOrder(int capacity) {  this.capacity = capacity;  items = new Item[capacity];  }  public Item getItem(int index){  try {  return items[index];  }catch (IndexOutOfBoundsException e) {  System.out.println("Out of bounds");  }  return null;  }  public int size(){  return size;  }  private void copy(int newCapacity){  Item[] newItems = new Item[newCapacity];  if (size >= 0) System.arraycopy(items, 0, newItems, 0, size);  items = newItems;  }  @Override  public boolean add(Item item) {  if (size == capacity - 1){  capacity \*= 2;  copy(capacity);  }  items[size] = item;  size+=1;  return true;  }  @Override  public boolean remove(String name) {  for (int i = 0; i < size - 2; i++)  if (Objects.equals(items[i].getTitle(), name))  for (int j = i; j < size - 2; j++) {  items[j] = items[j + 1];  size -= 1;  return true;  }  if (Objects.equals(items[size - 1].getTitle(), name)) {  items[size - 1] = null;  return true;  }  return false;  }  @Override  public int removeAll(String name) {  int s = 0;  boolean f = true;  do {  f = false;  for (int i = 0; i < size - 1; i++) {  if (Objects.equals(items[i].getTitle(), name)) {  f = true;  remove(name);  size--;  s++;  }  }  } while (f);  return s;  }  @Override  public int totalCount() {  return size;  }  @Override  public Item[] getArray() {  return items;  }  @Override  public int totalCost() {  int s = 0;  for (int i = 0; i < items.length; i++) {  s += items[i].getPrice();  }  return s;  }  @Override  public int numberOf(String name) {  int s = 0;  for (Item item : items) {  if (Objects.equals(item.getTitle(), name)) s++;  }  return s;  }  @Override  public Item[] getUnicArray() {  int k = 0;  Item[] temp = new Item[capacity];  for (int i = 0; i < size - 2; i++) {  boolean f = false;  for (int j = i + 1; j < size - 1; j++) {  if (items[i] == items[j]) {  f = true;  break;  }  }  if (!f) {  temp[k] = items[i];  k++;  }  }  return temp;  }  @Override  public Item[] getSortArray() {  return new Item[0];  }  } |

#### Класс «OrderManager»

Хранит данные о заказе с помощью ассоциативного массива

Листинг 88 - Код класса OrderManager

|  |
| --- |
| import java.util.GregorianCalendar;  import java.util.HashMap;  public class OrderManager {  private RestaurantOrder[] orders;  private int capacity;  private int size = 0;  private HashMap<String , RestaurantOrder> hashMap = new HashMap<>();  public OrderManager(int capacity) {  this.capacity = capacity;  orders = new RestaurantOrder[capacity];  }  public void addOrder(RestaurantOrder order){  if (size == capacity - 1){  capacity \*= 2;  copy(capacity);  }  orders[size] = order;  size+=1;  }  public RestaurantOrder getOrder(int index) {  try {  return orders[index];  }catch (IndexOutOfBoundsException e) {  System.out.println("Out of bounds");  }  return null;  }  public int getSize() {  return size;  }  private void copy(int newCapacity){  RestaurantOrder[] newOrders = new RestaurantOrder[newCapacity];  if (size >= 0) System.arraycopy(orders, 0, newOrders, 0, size);  orders = newOrders;  }  public void remove(RestaurantOrder order) {  for (int i = 0; i < size; i++)  if (orders[i] == order)  for (int j = i; j < size; j++)  orders[j] = orders[j - 1];  size -= 1;  }  public boolean addHash(String address, RestaurantOrder restaurantOrder) {  hashMap.put(address, restaurantOrder);  return true;  }  public Order getHash(String address) {  return hashMap.get(address);  }  public boolean removeHash(String address) {  hashMap.remove(address);  return true;  }  public boolean addItemHash(String string, Item item) {  RestaurantOrder restaurantOrder = new RestaurantOrder(1);  restaurantOrder.add(item);  hashMap.put(string, restaurantOrder);  return true;  }  public RestaurantOrder[] getArrayHash() {  int i = 0;  RestaurantOrder[] restaurantOrders = new RestaurantOrder[hashMap.size()];  for (RestaurantOrder restaurantOrder : hashMap.values()) {  restaurantOrders[i] = restaurantOrder;  i++;  }  return restaurantOrders;  }  public int totalCost() {  int s = 0;  RestaurantOrder[] restaurantOrders = getArrayHash();  for (int i = 0; i < restaurantOrders.length; i++) {  s += restaurantOrders[i].totalCost();  }  return s;  }  public int getCountHash(Item item) {  int s = 0;  Order[] orders = getArrayHash();  for(int i = 0; i < orders.length; i++) {  s += orders[i].numberOf(item.getTitle());  }  return s;  }  } |

#### Класс «Table»

Связывает столики с заказами и выдаёт ошибку, если столик уже связан с заказом

Листинг 89 - Код класса

|  |
| --- |
| import java.util.HashMap;  import java.util.Objects;  public class Table {  private int count = 0;  private int[] allIndex;  private int index;  private OrderManager orderManager;  private HashMap<Integer, String> hashMap;  public void connect(int index, String address) throws Exception {  boolean f = false;  for (int i = 0; i < allIndex.length; i++) {  if (allIndex[i] == index) {  f = true;  throw new Exception("IllegalTableNumber");  }  }  if (!f) {  for (String temp : hashMap.values()) {  if (Objects.equals(address, temp)) {  f = true;  break;  }  }  for (int temp : hashMap.keySet()) {  if (index == temp) {  f = true;  break;  }  }  if (f) throw new Exception("OrderAlreadyAddedException");  else hashMap.put(index, address);  int[] temp = new int[count];  System.arraycopy(allIndex, 0, temp, 0, allIndex.length);  allIndex = new int[count + 1];  System.arraycopy(temp, 0, allIndex, 0, temp.length);  allIndex[count + 1] = index;  count++;  }  }  } |

#### Класс «Dish»

Выдаёт ошибку, если блюдо бесплатное, не имеет имени или описания

Листинг 90 - Код класса Dish

|  |
| --- |
| public class Dish implements Item{  private final String title;  private final String description;  private final int price;  public Dish(String title, String description, int price) {  if (price < 0 || description.isEmpty() || title.isEmpty()) throw new IllegalArgumentException();  this.title = title;  this.description = description;  this.price = price;  }  public Dish(String title, String description) {  if (description.isEmpty() || title.isEmpty()) throw new IllegalArgumentException();  this.title = title;  this.description = description;  price = 0;  }  @Override  public int getPrice() {  return price;  }  @Override  public String getTitle() {  return title;  }  @Override  public String getDescription() {  return description;  }  } |

#### Класс «Drink»

Выдаёт ошибку, если напиток бесплатен, не имеет имени или описания

Листинг 91 - Код класса Drink

|  |
| --- |
| public class Drink implements Item{  private final String title;  private final String description;  private final int price;  public Drink(String title, String description, int price) {  if (price < 0 || description.isEmpty() || title.isEmpty()) throw new IllegalArgumentException();  this.title = title;  this.description = description;  this.price = price;  }  public Drink(String title, String description) {  if (description.isEmpty() || title.isEmpty()) throw new IllegalArgumentException();  this.title = title;  this.description = description;  price = 0;  }  public String getTitle() {  return title;  }  public String getDescription() {  return description;  }  public int getPrice() {  return price;  }  } |

## Выводы по работе:

Переделал практическую работу №16

# Практическая работа № 18

## Цель работы

Целью работы является изучение работы циклов, ввода/вывода, создание массивов и методов в Java.

## Теоретическое введение

Язык Java — это объектно-ориентированный язык программирования.

Программы написанные на Java могут выполняться на различных операционных системах при наличии необходимого ПО - Java Runtime Environment.

Чтобы объявить переменную, необходимо указать тип переменной и её имя.

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

Условие — это конструкция, позволяющая выполнять то или другое действие, в зависимости от логического значения, указанного в условии.

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

Для ввода данных используется класс Scanner из библиотеки пакетов.

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.

## Выполнение лабораторной работы

### Задание:

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

### Решение:

#### Класс «RandomArray»

Листинг 95 - Код "RandomArray"

Программа создаёт случайный массив чисел и сортирует его

|  |
| --- |
| import java.util.Arrays;  import java.util.Random;  import java.util.Scanner;  public class RandomArray {  public static void main(String[] args) {  int n;  System.out.print("Enter array size: ");  Scanner sc = new Scanner(System.in);  n = sc.nextInt();  Random rn = new Random();  int[] a = new int[n];  System.out.println("Original");  for (int i = 0; i < n; i++) {  a[i] = rn.nextInt(1, 30);  System.out.print(a[i]+" ");  }  System.out.println();  Arrays.sort(a);  System.out.println("Sorted");  for (int i = 0; i < n; i++) {  System.out.print(a[i]+" ");  }  }  } |

#### Результат работы класса «RandomArray» представлен на рисунке 7.

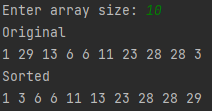


Рисунок 41 - Результат работы класса «RandomArray»

## Выводы по работе:

Изучил работу циклов и создание массива случайных чисел в Java.

# Практическая работа № 19

## Цель работы

Научиться создавать классы.

## Теоретическое введение

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.

Полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

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

## Выполнение лабораторной работы

### Задание:

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

### Решение:

#### Класс «Book»

Листинг 96 - Код "Book"

Класс принимает автора, название книги, год написания и количество страниц

|  |
| --- |
| package V3;  public class Book {  private String author, name;  private int year = 1999, pages = 100;  public Book(String author, String name, int year, int pages) {  this.author = author;  this.name = name;  this.year = year;  this.pages = pages;  }  public Book(String author, String name, int year) {  this.author = author;  this.name = name;  this.year = year;  }  public Book(String author, String name) {  this.author = author;  this.name = name;  }  public String getName() {  return name;  }  public int getPages() {  return pages;  }  public int getYear() {  return year;  }  public String getAuthor() {  return author;  }  public void setAuthor(String author) {  this.author = author;  }  public void setName(String name) {  this.name = name;  }  public void setPages(int pages) {  this.pages = pages;  }  public void setYear(int year) {  this.year = year;  }  @Override  public String toString() {  return "Book {" +  "author='" + author + '\'' +  ", name='" + name + '\'' +  ", year=" + year +  ", pages=" + pages +  '}';  }  } |

#### Класс-тестер «TestBook»

Листинг 97 - Код тестер "TestBook"

Проверка работы класса

|  |
| --- |
| package V3;  public class TestBook {  public static void main(String[] args) {  Book b1 = new Book("John Smith", "Atlantis", 2005, 520);  Book b2 = new Book("John Smith", "Atlantis");  System.out.println("Pages: " + b2.getPages());  b2.setName("Capital");  System.out.println(b2 + "\n" + b1);  }  } |

#### Результат тестирования класса «Book» представлен на рисунке 8

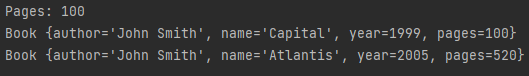


Рисунок 42 - Результат тестирования класса «Book»

## Выводы по работе:

Научился создавать классы, методы и назначать уровень доступа к методам/переменным

# Практическая работа № 20

## Цель работы

Изучить наследование в Java

## Теоретическое введение

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.

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово extends, после которого идет имя базового класса.

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

Можно запретить наследование с помощью ключевого слова final.

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.

Кроме обычных методов абстрактный класс может содержать абстрактные методы. Такие методы определяются с помощью ключевого слова abstract и не имеют никакого функционала. Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе.

## Выполнение лабораторной работы

### Задание:

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

### Решение:

#### Абстрактный класс «Furniture»

Листинг 98 - Код "Furniture"

Класс хранит основные методы для классов, которые будут его наследовать. Также он принимает следующие данные: материал, производитель, количество, цена, ширина, высота, глубина и вес.

|  |
| --- |
| abstract class Furniture {  protected String material, manufacturer;  protected int count;  protected double price, width, height, depth, weight;  public Furniture() {  material = "Wood";  manufacturer = "IKEA";  price = 1000;  count = 1;  width = 100;  height = 50;  depth = 100;  weight = 20;  }  public double getWidth() {  return width;  }  public void setWidth(double width) {  this.width = width;  }  public double getHeight() {  return height;  }  public void setHeight(double height) {  this.height = height;  }  public double getWeight() {  return weight;  }  public void setWeight(double weight) {  this.weight = weight;  }  public double getDepth() {  return depth;  }  public void setDepth(double depth) {  this.depth = depth;  }  public String getManufacturer() {  return manufacturer;  }  public void setManufacturer(String manufacturer) {  this.manufacturer = manufacturer;  }  public String getMaterial() {  return material;  }  public void setMaterial(String material) {  this.material = material;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  public int getCount() {  return count;  }  public void setCount(int count) {  this.count = count;  }  public abstract String toString();  } |

#### Класс «Table»

Листинг 99 - Код "Table"

Класс наследуется от «Furniture», добавляя следующие данные: количество ножек и угловой ли стол

|  |
| --- |
| public class Table extends Furniture {  private int legs;  private boolean isAngled;  public Table() {  legs = 4;  isAngled = true;  }  public Table(int l, boolean isA) {  legs = l;  isAngled = isA;  }  public int getLegs() {  return legs;  }  public boolean isAngled() {  return isAngled;  }  public void setAngled(boolean angled) {  isAngled = angled;  }  public void setLegs(int legs) {  this.legs = legs;  }  @Override  public String toString() {  return "Table {" +  "legs=" + legs +  ", width=" + width +  ", height=" + height +  ", depth=" + depth +  ", weight=" + weight +  ", angled=" + isAngled +  ", material='" + material + '\'' +  ", manufacturer='" + manufacturer + '\'' +  ", count=" + count +  ", price=" + price +  '}';  }  } |

#### Класс «Chair»

Листинг 100 - Код "Chair"

Класс наследуется от «Furniture», добавляя следующие данные: количество ножек, максимальный вес, материал самого сиденья и есть ли колёсики

|  |
| --- |
| public class Chair extends Furniture {  private int legs, maxWeight;  private String seatMaterial;  private boolean isRolling;  public Chair() {  legs = 5;  maxWeight = 150;  seatMaterial = "Leather";  isRolling = true;  }  public Chair(int l, int maxW, String sm, boolean isR) {  legs = l;  maxWeight = maxW;  seatMaterial = sm;  isRolling = isR;  }  public int getLegs() {  return legs;  }  public void setLegs(int legs) {  this.legs = legs;  }  public int getMaxWeight() {  return maxWeight;  }  public void setMaxWeight(int maxWeight) {  this.maxWeight = maxWeight;  }  public String getSeatMaterial() {  return seatMaterial;  }  public void setSeatMaterial(String seatMaterial) {  this.seatMaterial = seatMaterial;  }  public boolean isRolling() {  return isRolling;  }  public void setRolling(boolean rolling) {  isRolling = rolling;  }  @Override  public String toString() {  return "Chair {" +  "legs=" + legs +  ", isRolling=" + isRolling +  ", maxWeight=" + maxWeight +  ", main material='" + material + '\'' +  ", seat material='" + seatMaterial + '\'' +  ", manufacturer='" + manufacturer + '\'' +  ", count=" + count +  ", price=" + price +  ", width=" + width +  ", height=" + height +  ", depth=" + depth +  ", weight=" + weight +  '}';  }  } |

#### Класс «Sofa»

Листинг 101 - Код "Sofa"

Класс наследуется от «Furniture», добавляя следующие данные: количество дверок и секций, а также есть ли вешалка

|  |
| --- |
| public class Sofa extends Furniture {  private int doors, sections;  private boolean hangBar;  public Sofa() {  doors = 2;  sections = 4;  hangBar = true;  }  public Sofa(int d, int s, boolean hB) {  doors = d;  sections = s;  hangBar = hB;  }  public int getDoors() {  return doors;  }  public void setDoors(int doors) {  this.doors = doors;  }  public int getSections() {  return sections;  }  public void setSections(int sections) {  this.sections = sections;  }  public boolean isHangBar() {  return hangBar;  }  public void setHangBar(boolean hangBar) {  this.hangBar = hangBar;  }  @Override  public String toString() {  return "Sofa {" +  "doors=" + doors +  ", sections=" + sections +  ", hangBar=" + hangBar +  ", material='" + material + '\'' +  ", manufacturer='" + manufacturer + '\'' +  ", count=" + count +  ", price=" + price +  ", width=" + width +  ", height=" + height +  ", depth=" + depth +  ", weight=" + weight +  '}';  }  } |

#### Класс «FurnitureShop»

Листинг 102 - Код "FurnitureShop"

Проверка работы классов

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  import java.util.Scanner;  public class FurnitureShop {  public static class Defaults {  public double width = 100, height = 50, depth = 100, price = 1000;  public String manu = "IKEA", material = "Wood";  public int count = 1;  @Override  public String toString() {  return "Defaults {" +  "width=" + width +  ", height=" + height +  ", depth=" + depth +  ", material='" + material + '\'' +  ", manu='" + manu + '\'' +  ", count=" + count +  ", price=" + price +  '}';  }  }  public static void main(String[] args) {  Defaults dp = new Defaults(); //Для редактирования значений, которые в классе абстракции  List<Furniture> fn = new ArrayList<Furniture>();  Scanner sc = new Scanner(System.in);  String a; //а - выбор, n - для проверки на размер  while (true) {  System.out.println("1) Add new");  System.out.println("2) Show");  System.out.println("3) Change defaults");  System.out.println("4) Exit");  System.out.print("Choose action: ");  a = sc.next();  switch (a) {  case "1" -> {  while (a != "4")  a = AddFurniture(fn, sc, dp); //Костыль, чтобы создать бесконечный цикл для добавления  }  case "2" -> {  ShowFurniture(fn); //Вывод мебели  }  case "3" -> {  ChangeDefaults(dp, sc); //Изменить значения абстракции  }  case "4" -> {  System.exit(0); //Выход  }  }  }  }  private static void ChangeDefaults(Defaults dp, Scanner sc) {  System.out.println(dp);  System.out.print("Width: ");  dp.width = sc.nextDouble();  System.out.print("Height: ");  dp.height = sc.nextDouble();  System.out.print("Depth: ");  dp.depth = sc.nextDouble();  System.out.print("Material: ");  dp.material = sc.next();  System.out.print("Manu: ");  dp.manu = sc.next();  System.out.print("Count: ");  dp.count = sc.nextInt();  System.out.print("Price: ");  dp.price = sc.nextDouble();  }  private static void AddTable(List<Furniture> fn, Scanner sc) {  System.out.println("Legs(number), is angled(1-T, 0-F)");  fn.add(new Table(sc.nextInt(), sc.nextInt() == 1));  }  private static void AddChair(List<Furniture> fn, Scanner sc) {  System.out.println("Legs(n>2), max weight(n>50), seat material, is rolling(1-T, 0-F)");  fn.add(new Chair(sc.nextInt(), sc.nextInt(), sc.next(), sc.nextInt() == 1));  }  private static void AddSofa(List<Furniture> fn, Scanner sc) {  System.out.println("Doors(number), sections(number), has hang bar(1-T, 0-F)");  fn.add(new Sofa(sc.nextInt(), sc.nextInt(), sc.nextInt() == 1));  }  private static void ShowFurniture(List<Furniture> fn) {  for (Furniture furniture : fn) {  System.out.println(furniture);  }  }  static String AddFurniture(List<Furniture> fn, Scanner sc, Defaults dp) {  System.out.println("1) New table");  System.out.println("2) New chair");  System.out.println("3) New sofa");  System.out.println("4) Back");  System.out.print("Choose action: ");  switch (sc.next()) {  case "1" -> {  AddTable(fn, sc);  ApplyDefaults(fn.get(fn.size() - 1), dp);  }  case "2" -> {  AddChair(fn, sc);  ApplyDefaults(fn.get(fn.size() - 1), dp);  }  case "3" -> {  AddSofa(fn, sc);  ApplyDefaults(fn.get(fn.size() - 1), dp);  }  case "4" -> {  return "4";  }  }  return "1";  }  private static void ApplyDefaults(Furniture fn, Defaults dp) {  fn.setWidth(dp.width);  fn.setHeight(dp.height);  fn.setDepth(dp.depth);  fn.setMaterial(dp.material);  fn.setManufacturer(dp.manu);  fn.setCount(dp.count);  fn.setPrice(dp.price);  }  } |

#### Результат:

Главное меню представлено на рисунке 9

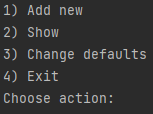


Рисунок 43 - Главное меню

Выбор добавляемого объекта представлен на рисунке 10

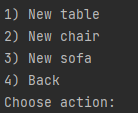


Рисунок 44 - Выбор добавления

Добавление стола представлено на рисунке 11

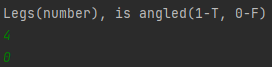


Рисунок 45 - Добавляем стол

Добавление стула представлено на рисунке 12

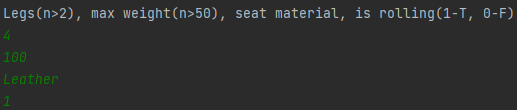


Рисунок 46 - Добавляем стул

Просмотр всех объектов представлен на рисунке 13

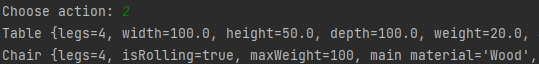


Рисунок 47 - Просмотр объектов

Изменение стандартных (абстрактных) значений представлено на рисунке 14

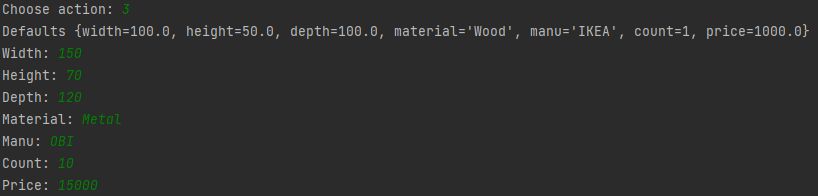


Рисунок 48 - Изменение стандартных значений

## Выводы по работе:

Изучил работу наследования и абстракции в Java.

# Практическая работа № 21

## Цель работы

Изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

## Теоретическое введение

Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы.

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

При этом надо учитывать, что если класс применяет интерфейс, то он должен реализовать все методы интерфейса.

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

В тоже время мы не можем напрямую создавать объекты интерфейсов.

## 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:

#### Интерфейс «Priceable»

Листинг 103 - Код "Priceable"

|  |
| --- |
| public interface Priceable {  double getPrice();  } |

#### Класс «Electronics»

Листинг 104 - Код "Electronics"

Класс реализует интерфейс «Priceable», принимая наименование, спецификацию и цену товара

|  |
| --- |
| public class Electronics implements Priceable{  private String name, specs;  private double price;  public Electronics(String name, String specs, double price){  this.name = name;  this.specs = specs;  this.price = price;  }  public void setPrice(double price) {  this.price = price;  }  public void setName(String name) {  this.name = name;  }  public void setSpecs(String specs) {  this.specs = specs;  }  public String getName() {  return name;  }  public String getSpecs() {  return specs;  }  @Override  public double getPrice() {  return price;  }  } |

#### Класс «Stationery»

Листинг 105 - Код "Stationery"

Класс реализует интерфейс «Priceable», принимая наименование, цену и количество товара

|  |
| --- |
| public class Stationery implements Priceable {  private String name;  private double price;  private int count;  public Stationery(String name, double price, int count) {  this.name = name;  this.price = price;  this.count = count;  }  public void setName(String name) {  this.name = name;  }  public void setPrice(double price) {  this.price = price;  }  public void setCount(int count) {  this.count = count;  }  public String getName() {  return name;  }  public int getCount() {  return count;  }  @Override  public double getPrice() {  return price;  }  } |

#### Класс «Tester»

Листинг 106 – Код «Tester»

Проверка работы классов

|  |
| --- |
| public class Tester {  public static void main(String[] args) {  Electronics el = new Electronics("PC", "i7 8700, gtx 1070", 70000);  Stationery st = new Stationery("Pencil", 10, 100);  System.out.println("Electronics: " + el.getPrice());  System.out.println("Stationery: " + st.getCount() + " for: " + st.getPrice());  }  } |

#### Результат:



Рисунок 49 - Результат Tester

## Выводы по работе:

Изучил понятие интерфейса, научился создавать интерфейсы в Java и применять их в программах.

# Практическая работа № 22

## Цель работы

Научиться создавать графический интерфейс пользователя, освоить на практике работу с различными объектами для создания ГИП, менеджерами размещения компонентов.

## Теоретическое введение

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.

TextField - текстовое поле или поля для ввода текста (можно ввести только одну строку).

Компонент TextArea похож на TextField, но в него можно вводить более одной строки.

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

### Менеджер BorderLayout:

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

### Менеджер GridLayout:

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

### 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)).

### 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.

## Выполнение лабораторной работы

### Задание:

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

### Решение:

#### Код «AnimBall»

Листинг 107 – Код «AnimBall»

Создание формы и анимации с помощью таймера и смены контента у элемента JLabel

|  |
| --- |
| import javax.swing.\*;  import java.awt.\*;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.util.Objects;  public class AnimBall extends JFrame {  int iFrame = 1;  ImageIcon img1 = new ImageIcon(Objects.requireNonNull(getClass().getResource("1.png")));  ImageIcon img2 = new ImageIcon(Objects.requireNonNull(getClass().getResource("2.png")));  ImageIcon img3 = new ImageIcon(Objects.requireNonNull(getClass().getResource("3.png")));  JPanel panel = new JPanel(new BorderLayout());  JLabel image = new JLabel(new ImageIcon("1.png"));  AnimBall() {  super("Example");  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setSize(350, 350);  add(image);  Timer timer = new Timer(1000, new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  switch (iFrame) {  case 1:  image.setIcon(img2);  break;  case 2:  image.setIcon(img3);  break;  case 3:  image.setIcon(img1);  iFrame = 0;  break;  }  iFrame++;  }  });  timer.start();  setLocationRelativeTo(null);  setVisible(true);  }  public static void main(String[] args) {  new AnimBall();  }  } |

#### Результат

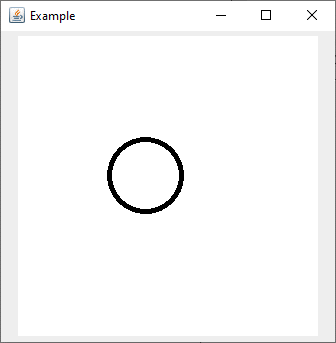


Рисунок 50 - Первый кадр

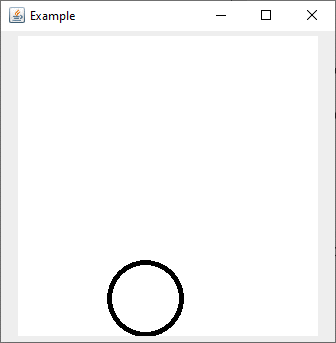


Рисунок 51 - Второй кадр

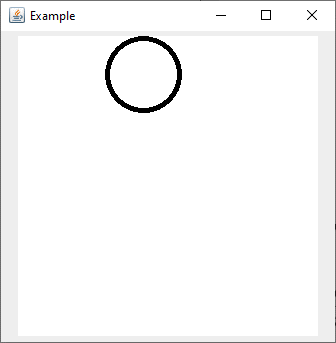


Рисунок 52 - Третий кадр

## Выводы по работе:

Научился создавать графический интерфейс пользователя, освоил на практике работу с различными объектами для создания ГИП, менеджерами размещения компонентов.

# Практическая работа № 23

## Цель работы

## Теоретическое введение

## Выполнение лабораторной работы

### Задание:

### Решение:

#### Результат

## Выводы по работе:

# Практическая работа № 24

## Цель работы

## Теоретическое введение

## Выполнение лабораторной работы

### Задание:

### Решение:

#### Результат

## Выводы по работе:

## Используемая литература

Список литературы тоже должен быть оформлен по ГОСТ. Первым пунктом можно указать:

1. Конспект лекций по дисциплине «Программирование на языке Джава», РТУ МИРЭА, лектор – старший преподаватель Зорина Н.В.
2. Карпова, И.П. Базы данных: Учебное пособие / И.П. Карпова. –СПб.: Питер, 2013. – 240 с.
3. Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств – Питер: 2016г. Режим доступа свободный: https://www.htbook.ru/kompjutery\_i\_seti/setevye\_tekhnologii/html5-i-css3-razrabotka-sajtov-dlja-ljubykh-brauzerov-i-ustrojstv;
4. Справочник по языку PHP [Электронный ресурс]:php.su— Режим доступа свободный: http://www.php.su;
5. И т.д.