Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Рязанский государственный радиотехнический

университет имени В. Ф. Уткина»

Кафедра ЭВМ

Отчёт о лабораторной работе №4

**«Наследование и полиморфизм»**

по дисциплине

«Промышленное программирование»

**Выполнили:**

ст. гр. 245 бригада № 6

Евдокимов Александр

Комарова Ксения

**Проверил:**

ст. пр. каф. ЭВМ Бастрычкин А.С.

Рязань 2024

# Цель работы

Изучение принципов ООП, приобретение навыков использования наследования и полиморфизма в Java-программах.

Практическая часть

# Задание 1

Создали приложение, удовлетворяющее требованиям:

* наследование применять только в тех заданиях, в которых это логически обосновано;
* аргументировать принадлежность классу каждого создаваемого метода;
* корректно переопределить для каждого класса методы equals(), hashCode(), toString().

Результат выполнения программы преведён на рисунке 1. Код программы представлен в приложениях A-D.

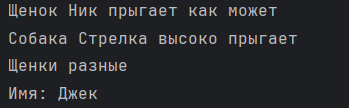


Рисунок 1 – Результат работы программы 1

# Задание 2

Создали консольное приложение, удовлетворяющее следующим требованиям:

* использовать возможности ООП: классы, наследование, полиморфизм, инкапсуляция;
* каждый класс должен иметь отражающее смысл название и информативный состав;
* наследование должно применяться только тогда, когда это имеет смысл;
* при кодировании должны быть использованы соглашения об оформлении кода Java Code Convention;
* классы должны быть грамотно разложены по пакетам;
* консольное меню должно быть минимальным;
* для хранения параметров инициализации можно использовать файлы.

Результат работы программы представлен на рисунках 2-7.

Код представлен в приложениях E-K.

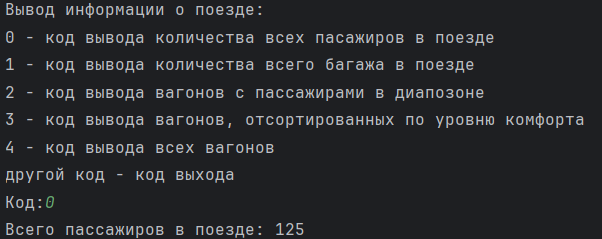


Рисунок 2 – Результат работы программы (команда 0)

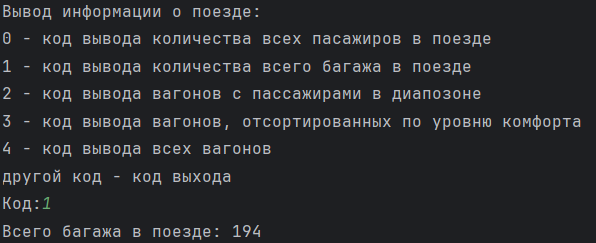


Рисунок 3 – Результат работы программы (команда 1)

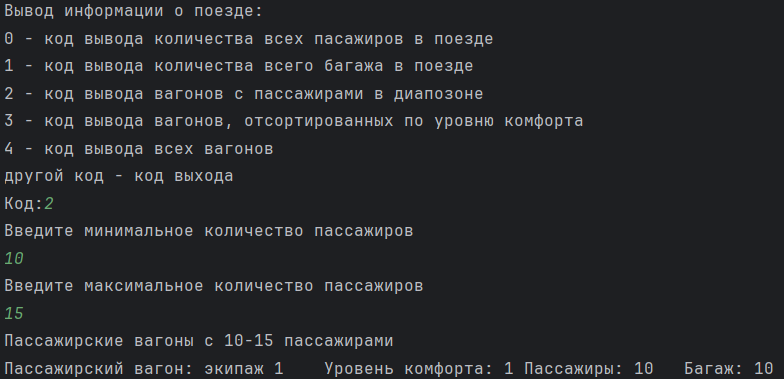


Рисунок 4 – Результат работы программы (команда 2)

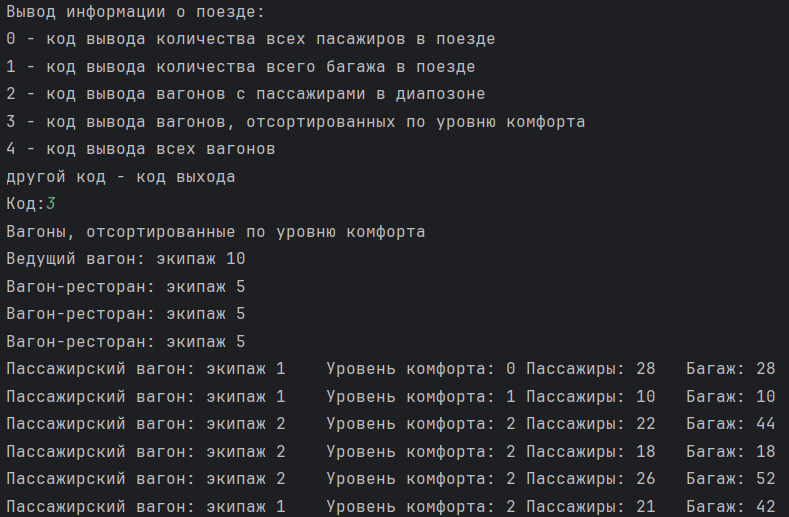


Рисунок 5 – Результат работы программы (команда 3)

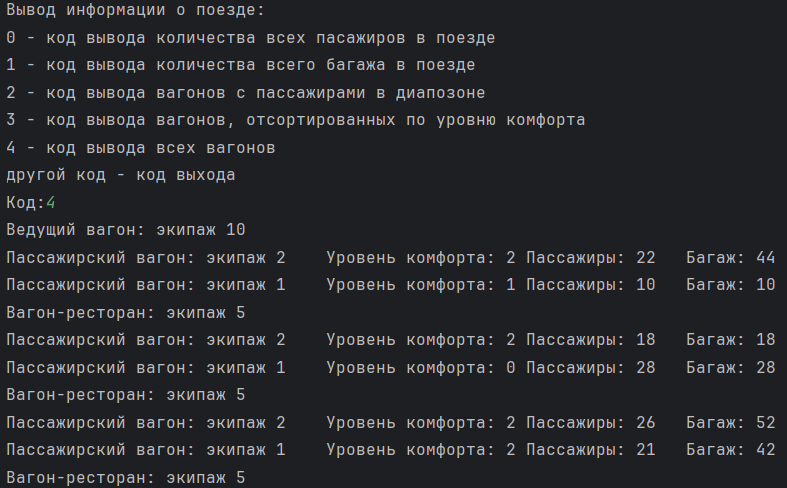


Рисунок 6 – Результат работы программы (команда 4)

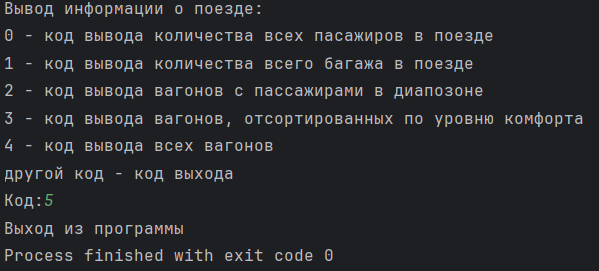


Рисунок 7 – Результат работы программы (иная команда)

**Заключение**

В ходе выполнения лабораторной работы были изучены принципы ООП, наследования и полиморфизма в Java.

Приложение А. Код класса Main

package Animal;  
  
public class Main {  
 public static void main(String[] args) {  
 Animal puppy = new Puppy("Ник");  
 Animal puppy2 = new Puppy("Джек");  
 puppy.jump();  
 Animal doggy = new Dog("Стрелка");  
 doggy.jump();  
 System.*out*.printf(puppy.equals(puppy2)?"Щенки одинаковы":"Щенки разные" + "\n");  
 puppy2.printName();  
 }  
}

Приложение B. Код класса Animal

package Animal;  
  
public abstract class Animal {  
 protected String name;  
  
 public Animal(String name){  
 this.name=name;  
 }  
  
 public Animal(){  
 name="";  
 }  
  
 public abstract void speak();  
  
 public abstract void jump();  
  
 public abstract void run();  
  
 public abstract void bite();  
  
 public abstract String toString();  
 public void printName(){  
 System.*out*.printf("Имя: %s\n", name);  
 }  
  
 public boolean equals(Animal animal) {  
 return this.name.equals(animal.name);  
 }  
  
 public int hashCode() {  
 return name.hashCode();  
 }  
}

Приложение C. Код класса Dog

package Animal;  
  
public class Dog extends Animal{  
 public Dog(String name){  
 super(name);  
 }  
  
 public Dog(){  
 super();  
 }  
  
 @Override  
 public void speak(){  
 System.*out*.printf("Собака %s громко гавкает\n", name);  
 }  
  
 @Override  
 public void jump(){ System.*out*.printf("Собака %s высоко прыгает\n", name); }  
  
 @Override  
 public void run(){  
 System.*out*.printf("Собака %s быстро бежит\n", name);  
 }  
  
 @Override  
 public void bite(){  
 System.*out*.printf("Собака %s сильно кусает\n", name);  
 }  
  
 @Override  
 public String toString(){  
 return name.isEmpty()?"Это собака":String.*format*("Это собака %s", name);  
 }  
}

Приложение D. Код класса Puppy

package Animal;  
  
public class Puppy extends Dog {  
 public Puppy(String name){  
 super(name);  
 }  
  
 public Puppy(){  
 super();  
 }  
  
 @Override  
 public void speak(){  
 System.*out*.printf("Щенок %s тихо лает\n", name);  
 }  
  
 @Override  
 public void jump(){  
 System.*out*.printf("Щенок %s прыгает как может\n", name);  
 }  
  
 @Override  
 public void run(){  
 System.*out*.printf("Щенок %s медленно бежит\n", name);  
 }  
  
 @Override  
 public void bite(){  
 System.*out*.printf("Щенок %s пытается укусить\n", name);  
 }  
  
 @Override  
 public String toString(){  
 return name.isEmpty()?"Это щенок":String.*format*("Это щенок %s", name);  
 }  
}

Приложение E. Код класса Main

package Transport;  
  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Main {  
 public static void main (String[] args){  
 Scanner in = new Scanner(System.*in*);  
 TrainGenerator trainGen = new TrainGenerator(10);  
 boolean isWorking = true;  
 int code;  
 while(isWorking){  
 System.*out*.print("""  
 Вывод информации о поезде:  
 0 - код вывода количества всех пасажиров в поезде  
 1 - код вывода количества всего багажа в поезде  
 2 - код вывода вагонов с пассажирами в диапозоне  
 3 - код вывода вагонов, отсортированных по уровню комфорта  
 4 - код вывода всех вагонов  
 другой код - код выхода  
 Код:""");  
 code = in.nextInt();  
 switch (code){  
 case 0:  
 System.*out*.printf("Всего пассажиров в поезде: %d\n", trainGen.getOverallPassengersCount());  
 break;  
 case 1:  
 System.*out*.printf("Всего багажа в поезде: %d\n", trainGen.getOverallLuggageCount());  
 break;  
 case 2:  
 System.*out*.println("Введите минимальное количество пассажиров");  
 int minPass = in.nextInt();  
 System.*out*.println("Введите максимальное количество пассажиров");  
 int maxPass = in.nextInt();  
 ArrayList<Carriage> found = trainGen.getCarriageWithPassengersBetween(minPass, maxPass);  
 if(found.isEmpty()) {  
 System.*out*.println("Ничего не найдено");  
 break;  
 }  
 System.*out*.printf("Пассажирские вагоны с %d-%d пассажирами\n", minPass, maxPass);  
 TrainGenerator.*print*(trainGen.getCarriageWithPassengersBetween(minPass, maxPass));  
 break;  
 case 3:  
 System.*out*.println("Вагоны, отсортированные по уровню комфорта");  
 TrainGenerator.*print*(trainGen.getTrainsSortedByComfort());  
 break;  
 case 4:  
 trainGen.printAll();  
 break;  
 default:  
 System.*out*.print("Выход из программы");  
 isWorking=false;  
 break;  
 }  
 }  
 in.close();  
 }  
}

Приложение F. Код класса Carriage

package Transport;  
  
public class Carriage {  
 protected int crewCount;  
  
 public Carriage(int crewCount){  
 this.crewCount=crewCount;  
 }  
  
 public Carriage(){  
 this.crewCount=0;  
 }  
  
 public int getCrewCount(){  
 return crewCount;  
 }  
  
 public String toString(){  
 return String.*format*("Вагон общего назначения: экипаж %d", crewCount);  
 }  
}

Приложение G. Код класса HeadCarriage

package Transport;  
  
class HeadCarriage extends Carriage {  
 public HeadCarriage(int crewCount){  
 super(crewCount);  
 }  
 public HeadCarriage(){ super(); }  
  
 @Override  
 public String toString(){  
 return String.*format*("Ведущий вагон: экипаж %d", crewCount);  
 }  
}

Приложение H. Код класса PassengerCarriage

package Transport;  
  
class PassengerCarriage extends Carriage{  
 private int comfortLevel;  
 private int passengersCount;  
 private int luggageCount;  
  
 public PassengerCarriage(int crewCount, int comfortLevel, int passengersCount, int luggageCount){  
 super(crewCount);  
 this.comfortLevel=comfortLevel;  
 this.passengersCount=passengersCount;  
 this.luggageCount=luggageCount;  
 }  
  
 public PassengerCarriage(){  
 super();  
 this.comfortLevel=0;  
 this.passengersCount=0;  
 this.luggageCount=0;  
 }  
  
 @Override  
 public String toString(){  
 return String.*format*("Пассажирский вагон: экипаж %d\tУровень комфорта: %d\tПассажиры: %d\t" +  
 "Багаж: %d", crewCount, comfortLevel, passengersCount, luggageCount);  
 }  
  
 public int getComfortLevel(){  
 return comfortLevel;  
 }  
  
 public int getPassengersCount() {  
 return passengersCount;  
 }  
  
 public int getLuggageCount() {  
 return luggageCount;  
 }  
}

Приложение J. Код класса RestaurantCarriage

package Transport;  
  
class RestaurantCarriage extends Carriage {  
 public RestaurantCarriage(int crewCount){  
 super(crewCount);  
 }  
 public RestaurantCarriage(){  
 super();  
 }  
  
 @Override  
 public String toString(){  
 return String.*format*("Вагон-ресторан: экипаж %d", crewCount);  
 }  
}

Приложение K. Код класса TrainGenerator

package Transport;  
import java.util.\*;  
  
public class TrainGenerator {  
  
 private ArrayList<Carriage> train;  
  
 TrainGenerator(int count){  
 Random rd = new Random();  
 train = new ArrayList<>();  
 train.add(new HeadCarriage(10));  
 for(int i =1; i<count; i++)  
 if(i%3==0) train.add(new RestaurantCarriage(5));  
 else {  
 int countedPassengers = rd.nextInt(10,30);  
 int countedLuggage = countedPassengers \* rd.nextInt(1,3);  
 train.add(new PassengerCarriage(rd.nextInt(1,3),  
 rd.nextInt(0,3), countedPassengers, countedLuggage));  
 }  
 }  
  
 public void printAll(){  
 for(Carriage each : train){  
 System.*out*.println(each);  
 }  
 }  
  
 public static void print(ArrayList<Carriage> train){  
 for(Carriage each : train){  
 System.*out*.println(each);  
 }  
 }  
  
 public int getOverallPassengersCount(){  
 int overallPassengersCount = 0;  
 for(Carriage each :train){  
 if(each instanceof PassengerCarriage){  
 overallPassengersCount+=((PassengerCarriage) each).getPassengersCount();  
 }  
 }  
 return overallPassengersCount;  
 }  
  
 public int getOverallLuggageCount(){  
 int overallLuggageCount = 0;  
 for(Carriage each :train)  
 if(each instanceof PassengerCarriage)  
 overallLuggageCount+=((PassengerCarriage) each).getLuggageCount();  
  
 return overallLuggageCount;  
 }  
  
 public ArrayList<Carriage> getCarriageWithPassengersBetween(int min, int max) {  
 ArrayList<Carriage> carriages = new ArrayList<>();  
 for (Carriage each :train) {  
 if (!(each instanceof PassengerCarriage)) continue;  
  
 if( ((PassengerCarriage) each).getPassengersCount() >= min &&  
 ((PassengerCarriage) each).getPassengersCount() <= max )  
 carriages.add(each);  
 }  
 return carriages;  
 }  
  
 public ArrayList<Carriage> getTrainsSortedByComfort() {  
  
 ArrayList<Carriage> sorted = (ArrayList<Carriage>)train.clone();  
  
 boolean swapped=true;  
 while(swapped) {  
 swapped = false;  
 for (int j = 0; j < sorted.size() - 1; j++) {  
 int comfortCurr = (sorted.get(j) instanceof PassengerCarriage)?((PassengerCarriage)sorted.get(j)).getComfortLevel():-1;  
 int comfortNext = (sorted.get(j+1) instanceof PassengerCarriage)?((PassengerCarriage)sorted.get(j+1)).getComfortLevel():-1;  
 if (comfortCurr > comfortNext) {  
 Collections.*swap*(sorted, j, j+1);  
 swapped = true;  
 }  
 }  
 }  
  
 return sorted;  
 }  
  
}