|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6) \_\_\_\_\_\_

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/12 Интеллектуальный анализ больших**

**данных в системах поддержки принятия решений**

**Отчет**

**по лабораторной работе № 4**

**Название:** Внутренние классы. Интерфейсы

**Дисциплина:** Языки программирования для работы с большими данными

Студент гр. ИУ6-23М **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_** Г. Е. Горский **\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**П.В. Степанов **\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

**Цель:** ознакомиться с базовыми принципами языка Java для работы с большими данными

**Вариант 1:**

1. Создать класс Cправочная Cлужба Oбщественного Tранспорта с внутренним классом, с помощью объектов которого можно хранить информацию о времени, линиях маршрутов и стоимости проезда.
2. Создать класс Computer (компьютер) с внутренним классом, с помощью объектов которого можно хранить информацию об операционной системе, процессоре и оперативной памяти.

**Решение:**

import java.util.ArrayList;  
import java.util.Date;  
public class Disp\_transport {  
  
 private class Inner\_Class {  
  
 private ArrayList<String> lines;  
   
 public String getDate() {  
 return date;  
 }  
 public void setDate(String date) {  
 this.date = date;  
 }  
  
 public ArrayList<String> getLines() {  
 return lines;  
 }  
  
 public void setLines(ArrayList<String> lines) {  
 this.lines = lines;  
 }  
  
 public ArrayList<Integer> getPrice() {  
 return price;  
 }  
  
 public void setPrice(ArrayList<Integer> price) {  
 this.price = price;  
 }  
  
 private String date;  
 private ArrayList<Integer> price;  
 public void date(){  
 Date getDateOfTask = new Date();  
 getDateOfTask.getDate();  
 this.setDate(String.valueOf(getDateOfTask));  
 System.out.println(" информация о времени " + this.getDate());  
 }  
 public void lines(){  
 ArrayList<String> line = new ArrayList<String>();  
 line.add("Москва-Анапа");  
 this.setLines(line);  
 System.out.println(" линии маршрутов " + this.getLines());  
 }  
 public void price(){  
 ArrayList<Integer> price = new ArrayList<Integer>();  
 price.add(20);  
 this.setPrice(price);  
 System.out.println(" стоимость проезда " + this.getPrice());  
 }  
 }  
 void display\_Inner() {  
 Inner\_Class inner = new Inner\_Class();  
 inner.date();  
 inner.lines();  
 inner.price();  
 }  
  
 public static void main(String[] args) {  
  
 Disp\_transport dispTransport = new Disp\_transport();  
 dispTransport.display\_Inner();  
 }  
}

Результат выполнения представлен на рисунке 1.

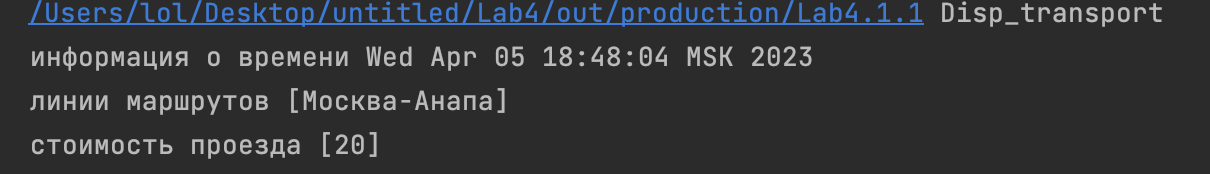
****

Рисунок 1 — пример выполнения

import java.util.ArrayList;  
public class Computer {  
 private class Inner\_Class {  
 private ArrayList<String> opSistem;  
 public ArrayList<String> getOpSistem() {  
 return opSistem;  
 }  
 public void setOpSistem(ArrayList<String> opSistem) {  
 this.opSistem = opSistem;  
 }  
 public ArrayList<String> getProcessor() {  
 return processor;  
 }  
 public void setProcessor(ArrayList<String> processor) {  
 this.processor = processor;  
 }  
 public ArrayList<Integer> getopmemory() {  
 return opmemory;  
 }  
 public void setopmemory(ArrayList<Integer> opmemory) {  
 this.opmemory = opmemory;  
 }  
 private ArrayList<String> processor;  
 private ArrayList<Integer> opmemory;  
 public void processor(){  
 ArrayList<String> proc = new ArrayList<String>();  
 proc.add("Intel core i5-12400f");  
 this.setProcessor(proc);  
 System.out.println(" процессор " + this.getProcessor());  
 }  
 public void opSistem(){  
 ArrayList<String> opsyst = new ArrayList<String>();  
 opsyst.add("Gentoo");  
 this.setOpSistem(opsyst);  
 System.out.println(" операционная система " + this.getOpSistem());  
 }  
 public void opmemory(){  
 ArrayList<Integer> opmemory = new ArrayList<Integer>();  
 opmemory.add(16);  
 this.setopmemory(opmemory);  
 System.out.println(" память " + this.getopmemory());  
 }  
 }  
 void display\_Inner() {  
 Inner\_Class inner = new Inner\_Class();  
 inner.processor();  
 inner.opSistem();  
 inner.opmemory();  
 }  
 public static void main(String[] args) {  
 Computer computer = new Computer();  
 computer.display\_Inner();  
 }  
}

Результат выполнения представлен на рисунке 2.

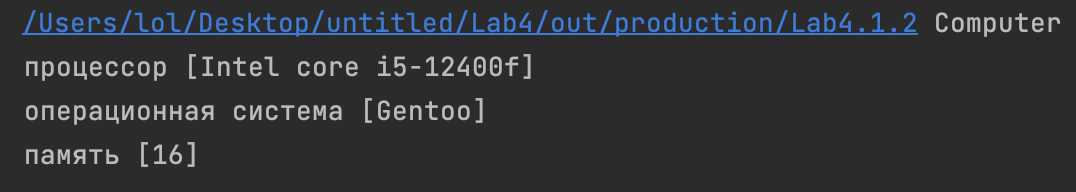


Рисунок 2 — пример выполнения

**Вариант 2:** реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов

1. interface Врач <- class Хирург <- class Нейрохирург.
2. interface Корабль <- class Грузовой Корабль <- class Танкер.

**Решение:**

public class Main {  
 public static void main(String[] args) {  
 Doctor doctor = new NeuroSurgeon("lol");  
 doctor.getProfessoin();  
 NeuroSurgeon neuroSurgeon = new NeuroSurgeon("lol");  
 neuroSurgeon.getNapravlenie();  
 }  
}  
interface Doctor{  
 void getProfessoin();  
}  
  
abstract class Surgeon implements Doctor{  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
 public Surgeon(String name) {  
 this.name = name;  
 }  
 abstract void getNapravlenie();  
 @Override  
 public void getProfessoin() {  
 System.out.print("interface Врач <- class Хирург ");  
 }  
}  
class NeuroSurgeon extends Surgeon{  
 public NeuroSurgeon(String name) {  
 super(name);  
 }  
 @Override  
 void getNapravlenie() {  
 System.out.print("<- Нейрохирург");  
 }  
}

Результат выполнения представлен на рисунке 3.

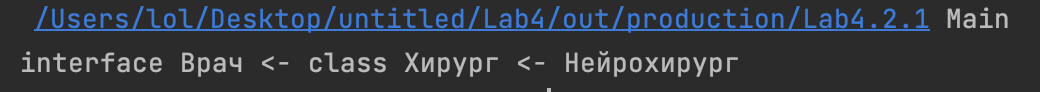


Рисунок 3 — пример выполнения

public class Main {  
 public static void main(String[] args) {  
 Ship ship = new Tanker("American");  
 ship.getShip();  
 Tanker tanker = new Tanker("Rus");  
 tanker.getCargoShip();  
 }  
}  
interface Ship{  
 void getShip();  
}  
  
abstract class CardoShip implements Ship{  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
  
 public CardoShip(String name) {  
 this.name = name;  
 }  
 abstract void getCargoShip();  
  
 @Override  
 public void getShip() {  
 System.out.print("interface Корабль <- class Грузовой Корабль ");  
 }  
}  
class Tanker extends CardoShip{  
 public Tanker(String name) {  
 super(name);  
 }  
 @Override  
 void getCargoShip(){  
 System.out.print("<- class Танкер.");  
 }  
}

Результат выполнения представлен на рисунке 4.

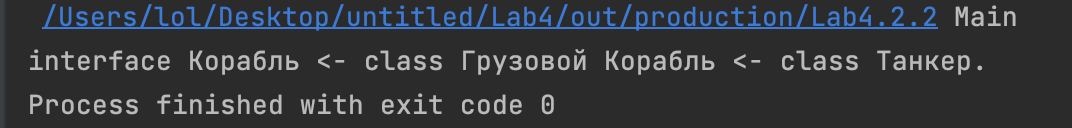


Рисунок 4 — пример выполнения

**Вывод:** были разработаны классы и методы согласно вариантам.