

Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

По дисциплине «Современные платформы программирования»

Выполнил:

Студент 3 курса

Группы ПО-3

Кабачук Д. С.

Проверил:

Крощенко А.А.

Брест 2020 г.

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Ход работы

Задача 1: Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

11) interface Устройство Печати ← class Принтер ← class Лазерный Принтер.

Код:

```
public class Main {  
    public static void main(String[] args) {  
        LaserPrinter laserPrinter = new LaserPrinter(300,true,7);  
        laserPrinter.print();  
    }  
}  
  
interface PrintingDevice {  
    void print();  
    int textLength();  
}  
  
class Printer implements PrintingDevice {  
    int maxTextLength;  
    boolean wifi;  
    Printer(int maxTextLength, boolean wifi){  
        this.wifi = wifi;  
        this.maxTextLength = maxTextLength;  
    }  
  
    public void print() {
```

```
        System.out.println("Максимальная длина строки: " +
maxTextLength + " \n" + "Наличие wi-fi: " + wifi + "\nТекущая длина
строки: " + textLength());

    }

    public int textLength() {

        return 1;

    }

}

class LaserPrinter extends Printer {

    private int maxTextLength;

    LaserPrinter(int maxTextLength, boolean wifi, int recordWeight) {

        super(maxTextLength, wifi);

        this.maxTextLength = maxTextLength;

    }

    public int textLength() {

        return 10;

    }

}
```

Результат:

```
Максимальная длина строки: 300
Наличие wi-fi: true
Текущая длина строки: 10
```

Задача 2: В следующих заданиях требуется создать суперкласс (абстрактный класс, интерфейс) и определить общие методы для данного класса. Создать подклассы, в которых добавить специфические свойства и методы. Часть методов переопределить. Создать массив объектов суперкласса и заполнить объектами подклассов. Объекты подклассов идентифицировать конструктором по имени или идентификационному номеру. Использовать объекты подклассов для моделирования реальных ситуаций и объектов.

Создать суперкласс Музыкальный инструмент и классы Ударный, Струнный, Духовой. Создать массив объектов Оркестр. Осуществить вывод состава оркестра.

Код:

```
public class Main {

    public static void main(String[] args) {

        Instrument instrument[] = new Instrument[3];

        instrument[0] = new Wind(5,120,10);

        instrument[1] = new Percussion(2, 800, 100);

        instrument[2] = new Stringed (6, 160, 7);

        System.out.println("Orchestra is: ");

        instrument[0].play(3, 0.7);

        instrument[1].play(5, 0.5);

        instrument[2].play(10, 0.8);

    }

}

class Percussion extends Instrument {

    int size;

    public Percussion (int size, double weight, double typeNumber){

        this.size = size;
```

```

        this.weight = weight;

        this.typeNumber = typeNumber;
    }

    @Override void play(double time, double speed){

        System.out.println("Percussion is playing " + time + "h with " +
speed + " speed");

    }

    void setSize(){

        this.size = 5;

    }
}

class Stringed extends Instrument {

    int stringsNumber;

    public Stringed (int stringsNumber, double weight, double
typeNumber){

        this.stringsNumber = stringsNumber;

        this.weight = weight;

        this.typeNumber = typeNumber;

    }

    @Override void play(double time, double speed){

        System.out.println("Stringed is playing " + time + "h with " +
speed + " speed");

    }

    void checkNumber(int stringsNumber){

        this.stringsNumber += stringsNumber;
    }
}

```

```

    }
}

abstract class Instrument {
    double weight;
    double typeNumber;

    abstract void play(double time, double speed);
}

class Wind extends Instrument {
    int value;

    public Wind (int value, double weight, double typeNumber){
        this.value = value;
        this.weight = weight;
        this.typeNumber = typeNumber;
    }

    @Override void play(double time, double speed){
        System.out.println("Wind is playing " + time + "h with " + speed
+ " speed");
    }

    void getValue(int value){
        this.value -= value;
    }
}

```

Результат:

```
Orchestra is:  
Wind is playing 3.0h with 0.7 speed  
Percussion is playing 5.0h with 0.5 speed  
Stringed is playing 10.0h with 0.8 speed
```

Задача 3: В задании 3 ЛР No4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

- 11) Система **Аэрофлот**. **Администратор** формирует летную **Бригаду** (пилоты, штурман, радист, стюардессы) на **Рейс**. Каждый **Рейс** выполняется **Самолетом** с определенной вместимостью и дальностью полета. **Рейс** может быть отменен из-за погодных условий в **Аэропорту** отлета или назначения. **Аэропорт** назначения может быть изменен в полете из-за технических неисправностей, о которых сообщил командир.

Код:

```
package com.company;  
  
import java.time.Instant;  
import java.time.LocalDateTime;  
import java.time.Month;  
import java.time.ZoneId;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class Main {  
    public static void main(String[] args) {  
        Aeroflot aeroflot1 = new Aeroflot("Minsk");  
        Aeroflot aeroflot2 = new Aeroflot("Monreale");  
        Aeroflot aeroflot3 = new Aeroflot("Rome");  
  
        List<Admin> admins = new ArrayList<>();  
        admins.add(new Admin("FirstAdmin", TypeOfEmployee.ADMIN));  
        aeroflot1.setAdmins(admins);  
  
        Plane plane1 = new Plane(TypeOfPlane.AVERAGE);  
        Admin firstAdmin = admins.get(0);  
        firstAdmin.getPosition();  
  
        List<CrewMember> crewMembers = firstAdmin.setCrewMembers (  
            new CrewMember("Andrey", TypeOfEmployee.PILOT),  
            new CrewMember("George", TypeOfEmployee.PILOT),
```

```

        new CrewMember("Michael", TypeOfEmployee.NAVIGATOR),
        new CrewMember("Tomas", TypeOfEmployee.OPERATOR),
        new CrewMember("Chloe", TypeOfEmployee.STEWARDESS),
        new CrewMember("Charley", TypeOfEmployee.STEWARDESS)
    );
    plane1.setCrewMembers(crewMembers);

    LocalDateTime localDate = LocalDateTime.of(2019,
Month.DECEMBER, 11, 22, 00);
    LocalDateTime departureDate = LocalDateTime.of(2019,
Month.DECEMBER, 10, 22, 00);
    Date destinationDate =
Date.from(Instant.from(localDate.atZone(ZoneId.systemDefault())));
    Date departureDateUTC =
Date.from(Instant.from(departureDate.atZone(ZoneId.systemDefault())));
    Flight flight1 = new Flight(
        "Monreale",
        "Minsk",
        departureDateUTC,
        destinationDate,
        plane1
    );

    aeroflot1.addFlight(flight1);
    aeroflot2.addFlight(flight1);

    System.out.println(aeroflot1);
    System.out.println(aeroflot2);
    System.out.println(aeroflot3);

    System.out.println("\n//////////////////////////////////////////
//////////////////////////////////////////\n");

    LocalDateTime localDate2 = LocalDateTime.of(2019,
Month.DECEMBER, 12, 22, 00);
    Date date2 =
Date.from(Instant.from(localDate2.atZone(ZoneId.systemDefault())));

    changeDestination(aeroflot2, aeroflot3, date2, flight1);

    System.out.println(aeroflot1);
    System.out.println(aeroflot2);
    System.out.println(aeroflot3);

    discardFlight(aeroflot1, aeroflot3, flight1);

    System.out.println("\n//////////////////////////////////////////
//////////////////////////////////////////\n");
    System.out.println(aeroflot1);
    System.out.println(aeroflot2);

```



```

        System.out.println(aeroflot3);
    }

    private static void discardFlight(
        Aeroflot destination,
        Aeroflot departure,
        Flight flight
    ) {
        if(flight.getDepartureTime().before(new Date())) {
            System.out.println("U can't discard flight. Change
destination");
            return;
        }
        if(destination.getFlights().contains(flight) &&
departure.getFlights().contains(flight)) {
            departure.discardFlight(flight);
            destination.discardFlight(flight);
        }
    }

    private static void changeDestination(
        Aeroflot oldDestination,
        Aeroflot newDestination,
        Date newDestinationTime,
        Flight flight
    ) {
        List<Flight> oldFlights = oldDestination.getFlights();
        int oldFlightIndex = oldFlights.indexOf(flight);
        Flight oldFlight = oldFlights.get(oldFlightIndex);

        if(oldFlight.getDestinationTime().before(new Date())) {
            System.out.println("U can't change destination");
            return;
        }

        flight.setDestination(newDestination.getName());
        flight.setDestinationTime(newDestinationTime);

        oldDestination.discardFlight(flight);
        newDestination.addFlight(flight);
        System.out.println("Destination is changed. Have a nice
flight");
    }
}

```

```

class Admin implements Employee {
    private static final AtomicInteger count = new AtomicInteger(1);
    private int id;
    private String name;
    private TypeOfEmployee type;
}

```

```

    public Admin(String name, TypeOfEmployee type) {
        this.id = count.incrementAndGet();
        this.name = name;
        this.type = type;
    }

    public List<CrewMember> setCrewMembers(
        CrewMember pilot1,
        CrewMember pilot2,
        CrewMember navigator,
        CrewMember operator,
        CrewMember stewardess1,
        CrewMember stewardess2
    ) {
        List<CrewMember> crew = new ArrayList<>();
        crew.add(pilot1);
        crew.add(pilot2);
        crew.add(navigator);
        crew.add(operator);
        crew.add(stewardess1);
        crew.add(stewardess2);
        return crew;
    }

    @Override
    public void getPosition() {
        System.out.println("I'm " + this.type);
    }

    @Override
    public String toString() {
        return "\n\t\t Admin {" +
            "\n\t\t\t id=" + id +
            ", \n\t\t\t name='" + name + '\'' +
            "\n\t\t }";
    }
}

class Aeroflot {
    private static final AtomicInteger count = new AtomicInteger(1);
    private int id;
    private String name;
    private List<Admin> admins;
    private List<Flight> flights;

    public Aeroflot(String name) {
        this.id = count.incrementAndGet();
        this.name = name;
        this.admins = new ArrayList<>();
        this.flights = new ArrayList<>();
    }
}

```

```

    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<Admin> getAdmins() {
        return admins;
    }

    public void setAdmins(List<Admin> admins) {
        this.admins = admins;
    }

    public List<Flight> getFlights() {
        return flights;
    }

    public void setFlights(List<Flight> flights) {
        this.flights = flights;
    }

    public void addFlight(Flight flight) {
        this.flights.add(flight);
    }

    public void addAdmin(Admin admin) {
        this.admins.add(admin);
    }

    public void discardFlight(Flight flight) {
        this.flights.remove(flight);
    }

    @Override
    public String toString() {
        return "Aeroflot {" +
            "\n\t id=" + id +

```

```

        ",\n\t name='" + name + '\'' +
        ",\n\t admins=" + admins +
        ",\n\t flights=" + flights +
        "\n}";
    }

}

class Flight {
    private String destination;
    private String departure;
    private Date destinationTime;
    private Date departureTime;
    private Plane plane;

    public Flight(String destination, String departure, Date
departureTime, Date destinationTime, Plane plane) {
        this.destination = destination;
        this.departure = departure;
        this.destinationTime = destinationTime;
        this.departureTime = departureTime;
        this.plane = plane;
    }

    public String getDestination() {
        return destination;
    }

    public void setDestination(String destination) {
        this.destination = destination;
    }

    public String getDeparture() {
        return departure;
    }

    public void setDeparture(String departure) {
        this.departure = departure;
    }

    public Date getDestinationTime() {
        return destinationTime;
    }

    public void setDestinationTime(Date destinationTime) {
        this.destinationTime = destinationTime;
    }

    public Date getDepartureTime() {
        return departureTime;
    }
}

```

```

    }

    public void setDepartureTime(Date departureTime) {
        this.departureTime = departureTime;
    }

    public Plane getPlane() {
        return plane;
    }

    public void setPlane(Plane plane) {
        this.plane = plane;
    }

    @Override
    public String toString() {
        return "\n\t\t Flight {" +
            "\n\t\t\t destination='" + destination + '\'' +
            "\n\t\t\t departure='" + departure + '\'' +
            "\n\t\t\t destinationTime=" + destinationTime +
            "\n\t\t\t departureTime=" + departureTime +
            "\n\t\t\t plane=" + plane +
            "\n\t}";
    }
}

class Plane {
    private static final AtomicInteger count = new AtomicInteger(1);
    private int id;
    private TypeOfPlane planeType;
    private List<CrewMember> crew;

    public Plane(TypeOfPlane planeType) {
        this.id = count.incrementAndGet();
        this.planeType = planeType;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public TypeOfPlane get.TypeOfPlane() {
        return planeType;
    }

    public void setTypeOfPlane(TypeOfPlane planeType) {
        this.planeType = planeType;
    }
}

```

```

    }

    public List<CrewMember> getCrew() {
        return crew;
    }

    public void setCrew(List<CrewMember> crew) {
        this.crew = crew;
    }

    public void setCrewMembers(List<CrewMember> crew) {
        this.crew = crew;
    }

    @Override
    public String toString() {
        return "\n\t\t\t\t\t Самолет {" +
            " \n\t\t\t\t\t\t\t id=" + id +
            ",\n\t\t\t\t\t\t\t Тип самолета=" + planeType +
            ",\n\t\t\t\t\t\t\t Персонал=" + crew +
            "\n\t\t\t\t\t}";
    }
}

interface Employee {
    public void getPosition();
}

enum TypeOfEmployee {
    ADMIN, PILOT, STEWARDESS, OPERATOR, NAVIGATOR;
}

enum TypeOfPlane {
    SMALL(), AVERAGE(), MAJOR();
}

class CrewMember {
    private String name;
    private TypeOfEmployee type;

    public CrewMember(String name, TypeOfEmployee type) {
        this.name = name;
        this.type = type;
    }
}

```

Результат 3:

```

I'm ADMIN
Aeroflot {
  id=2,
  name='Minsk',
  admins=[
    Admin {
      id=2,
      name='FirstAdmin'
    }
  ],
  flights=[
    Flight {,
      destination='Monreale',
      departure='Minsk',
      destinationTime=Wed Dec 11 22:00:00 MSK 2019,
      departureTime=Tue Dec 10 22:00:00 MSK 2019,
      plane=
        Plane{
          id=2,
          typeOfPlane=AVERAGE,
          crewMembers=[CrewMember{name='Andrey', type=PILOT}, CrewMember{name='George', type=PILOT}, CrewMember{name='Michael',
        }
      ]
    }
  ]
}
Aeroflot {
  id=3,
  name='Monreale',
  admins=[],
  flights=[
    Flight {,
      destination='Monreale',
      departure='Minsk',
      destinationTime=Wed Dec 11 22:00:00 MSK 2019,
      departureTime=Tue Dec 10 22:00:00 MSK 2019,
      plane=
        Plane{
          id=2,
          typeOfPlane=AVERAGE,
          crewMembers=[CrewMember{name='Andrey', type=PILOT}, CrewMember{name='George', type=PILOT}, CrewMember{name='Michael',
        }
      ]
    }
  ]
}
}

```

Сменить пункт назначения

```
U can't change destination
Aeroflot {
  id=2,
  name='Minsk',
  admins=[
    Admin {
      id=2,
      name='FirstAdmin'
    },
  ],
  flights=[
    Flight {,
      destination='Monreale',
      departure='Minsk',
      destinationTime=Wed Dec 11 22:00:00 MSK 2019,
      departureTime=Tue Dec 10 22:00:00 MSK 2019,
      plane=
        Plane{
          id=2,
          typeOfPlane=AVERAGE,
          crewMembers=[CrewMember{name='Andrey', type=PILOT}, CrewMember{name='George', type=PILOT},
        ]
      ]
    }
  ]
}
Aeroflot {
  id=3,
  name='Monreale',
  admins=[],
  flights=[
    Flight {,
      destination='Monreale',
      departure='Minsk',
      destinationTime=Wed Dec 11 22:00:00 MSK 2019,
      departureTime=Tue Dec 10 22:00:00 MSK 2019,
      plane=
        Plane{
          id=2,
          typeOfPlane=AVERAGE,
          crewMembers=[CrewMember{name='Andrey', type=PILOT}, CrewMember{name='George', type=PILOT},
        ]
      ]
    }
  ]
}
```


Удалить рейс

```
Aeroflot {
  id=2,
  name='Minsk',
  admins=[
    Admin {
      id=2,
      name='1'
    },
  ],
  flights=[]
}
Aeroflot {
  id=3,
  name='Monreale',
  admins=[],
  flights=[]
}
Aeroflot {
  id=4,
  name='Rome',
  admins=[],
  flights=[]
}
```

Вывод: в ходе выполненной работы были приобретены практические навыки в области объектно-ориентированного проектирования.