МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №5

Выполнила:

студентка 3 курса
группы ПО-9
Шубич Дарья
Константинова

Проверил:

Крощенко А.А.

Брест 2024

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

Вариант 11

Задание 1

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:
interface Устройство Печати ← class Принтер ← class Лазерный Принтер.
Входные данные:

```
LaserPrinter laserPrinter = new LaserPrinter("Samsung S90");
laserPrinter.on();
laserPrinter.print("Лабораторная работа №5, задание 1, выполнила Шубич Дарья Константиновна.");
laserPrinter.off();
```

Результат программы

```
Подключение лазерного принтера Samsung S90
Печать на принтере Samsung S90: Лабораторная работа №5, задание 1, выполнила Шубич Дарья Константиновна.
Отключение лазерного принтера Samsung S90
```

Код программы:

```
package org.example;

public interface PrintDevice {
    void print(String text);
}

public abstract class Printer implements PrintDevice {
    private String model;

    public Printer(String model) {
        this.model = model;
    }

    public String getModel() {
        return model;
    }

    public void print(String text) {
        System.out.println("Печать на принтере " + model + ": " + text );
    }

    // дополнительные абстрактные методы для реализации в подклассе

    public abstract void on();
    public abstract void off();
}


public class LaserPrinter  extends Printer{

    public LaserPrinter(String model) {
```

```
        super(model);
    }

    @Override
    public void on() {
        System.out.println("Подключение лазерного принтера " + getModel());
    }

    @Override
    public void off() {
        System.out.println("Отключение лазерного принтера " + getModel());
    }
}
```

Задание 2

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

```
Orchestra orchestra = new Orchestra();
MusicalInstrument[] instruments = new MusicalInstrument[5];

instruments[0] = new Drums("Electric Drums 432", 5);
instruments[1] = new Stringed("Bas Guitar Fender 400", 4);
instruments[2] = new Stringed("Acoustic Guitar Fender 212", 6);
instruments[3] = new Wind("Flute 'Flora'", "Дуб");
instruments[4] = new Wind("Saxophone 'JazzyVibes'", "Латунь");
for (MusicalInstrument instrument : instruments) {
    System.out.println(instrument.toString());
    instrument.tune();
    System.out.println();
}
for (MusicalInstrument instrument : instruments) {
    orchestra.addInstrument(instrument);
}
System.out.println("Оркестр:");
orchestra.printOrchestra();
```

Выходные данные:

```
Название: Electric Drums 432, Количество пэдов: 5
Настройка барабана: Electric Drums 432

Название: Bas Guitar Fender 400, Количество струн: 4
Настройка струнного инструмента: Bas Guitar Fender 400

Название: Acoustic Guitar Fender 212, Количество струн: 6
Настройка струнного инструмента: Acoustic Guitar Fender 212

Название: Flute 'Flora', Материал: Дуб
Настройка духового инструмента: Flute 'Flora'

Название: Saxophone 'JazzyVibes', Материал: Латунь
Настройка духового инструмента: Saxophone 'JazzyVibes'

Оркестр:
Electric Drums 432
Bas Guitar Fender 400
Acoustic Guitar Fender 212
Flute 'Flora'
Saxophone 'JazzyVibes'
```

Код программы:

```
package org.example;

public abstract class MusicalInstrument {
    private String name;

    public MusicalInstrument(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void play() {
        System.out.println("Играет: " + name);
    }
    @Override
    public String toString() {
        return "Название: " + name;
    }

    public abstract void tune();
}
```

```
public class Drums extends MusicalInstrument{
    private  int numberOfPads;

    public Drums(String name, int numberOfPads) {
        super(name);
        this.numberOfPads = numberOfPads;
    }

    public int getNumberOfPads() {
        return numberOfPads;
    }
    @Override
    public String toString() {
        return super.toString() + ", Количество пэдов: " + numberOfPads;
    }
    @Override
    public void tune() {
        System.out.println("Настройка барабана: " + getName());
    }
}
```

```
public class Stringed  extends MusicalInstrument{
    private int numberOfStringe;

    public Stringed(String name, int numberOfStringe) {
        super(name);
        this.numberOfStringe = numberOfStringe;
    }

    public int getNumberOfStringe() {
        return numberOfStringe;
    }

    @Override
    public void tune() {
```

```
        System.out.println("Настройка струнного инструмента: " +
getName());

    }
    @Override
    public String toString() {
        return super.toString() + ", Количество струн: " + numberOfStringe;
    }
}
```

```
public class Wind extends MusicalInstrument{
    private String material;

    public Wind(String name, String material) {
        super(name);
        this.material = material;
    }

    public String getMaterial() {
        return material;
    }
    @Override
    public String toString() {
        return super.toString() + ", Материал: " + material;
    }
    @Override
    public void tune() {
        System.out.println("Настройка духового инструмента: " + getName());

    }
```

```
public class Orchestra {
    private MusicalInstrument[] instruments;


    public Orchestra() {
        instruments = new MusicalInstrument[0];
    }

    public void addInstrument(MusicalInstrument instrument) {
        MusicalInstrument[] newArray = new
MusicalInstrument[instruments.length + 1];

        for (int i = 0; i < instruments.length; i++) {
            newArray[i] = instruments[i];
        }

        newArray[newArray.length - 1] = instrument;
        instruments = newArray;
    }

    public void printOrchestra(){
        for (MusicalInstrument instrument : instruments) {
            System.out.println(instrument.getName());
        }
    }
}
```

Задание 3

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

Aircraft и Airport не обладают поведением, которое можно было бы абстрагировать в виде интерфейсов или абстрактных классов. Они представляют собой простые модели данных, которые не имеют методов для реализации в подклассах. Таким образом, в данном случае замена объявлений суперклассов на абстрактные классы или интерфейсы не имеет смысла.

```
package org.example;

import java.util.ArrayList;
import java.util.List;

    class FlightCrew {

        public String getPosition() {
            return position;
        }

        private String name;
        private String position;

        public FlightCrew(String name, String position) {
            this.name = name;
            this.position = position;
        }

        public String getName() {
            return name;
        }

    }

    class Aircraft {
        private int seatingCapacity;
        private int flightRange;

        public Aircraft(int seatingCapacity, int flightRange) {
            this.seatingCapacity = seatingCapacity;
            this.flightRange = flightRange;
        }
    }

    class Flight {
        private String flightNumber;
        private Airport departureAirport;
        private Airport destinationAirport;
        private boolean isCanceled;
        private Aircraft aircraft;
        private List<FlightCrew> flightCrew;

        public Flight(String flightNumber, Airport departureAirport,
Airport destinationAirport, Aircraft aircraft) {
            this.flightNumber = flightNumber;
            this.departureAirport = departureAirport;
            this.destinationAirport = destinationAirport;
```

```
            this.aircraft = aircraft;
            this.isCanceled = false;
            this.flightCrew = new ArrayList<>();
        }

        public void addFlightCrewMember(FlightCrew crewMember) {
            flightCrew.add(crewMember);
        }

        public void setDestinationAirport(Airport newDestination) {
            this.destinationAirport = newDestination;
        }

        public void cancelFlight() {
            this.isCanceled = true;
        }

        public List<FlightCrew> getFlightCrew() {
            return flightCrew;
        }

        public Airport getDepartureAirport() {
            return departureAirport;
        }

        public String getFlightNumber() {
            return flightNumber;
        }

        public boolean isCanceled() {
            return isCanceled;
        }

        public void setCanceled(boolean canceled) {
            this.isCanceled = canceled;
        }

        public Airport setDestinationAirport() {
            return  destinationAirport;
        }

    }

    class Airport {
        private String name;
        private String location;

        public Airport(String name, String location) {
            this.name = name;
            this.location = location;
        }

        public String getName() {
            return name;

        }
    }

    class Administrator {
        public void formFlightCrew(Flight flight, List<FlightCrew>
crewMembers) {
```

```
            for (FlightCrew crewMember : crewMembers) {
                flight.addFlightCrewMember(crewMember);
            }
        }

        public void changeDestinationAirport(Flight flight, Airport
newDestination) {
            flight.setDestinationAirport(newDestination);
        }

        public void cancelFlight(Flight flight) {
            flight.cancelFlight();
        }
    }
```