МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчёт
по лабораторной работе №6

Выполнил:
студент группы ПО-9
Качаловский Данил Сергеевич

Проверил:
Крощенко А. А.

Брест 2024

Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Java

Вариант 7

Задание 1

Преподаватель. Класс должен обеспечивать одновременное взаимодействие с несколькими объектами класса Студент. Основные функции преподавателя – ПроверитьЛабораторнуюРаботу, ПровестиКонсультацию, ПринятьЭкзамен, ВыставитьОтметку, ПровестиЛекцию.

Код программы

```
package Lab6_1;
import java.util.ArrayList;
import java.util.List;

class Student {
    private String name;

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

    public String getName() {
        return name;
    }
}

class Teacher {
    private static Teacher instance;
    private List<Student> students;

    private Teacher() {
        students = new ArrayList<>();
    }

    public static Teacher getInstance() {
        if (instance == null) {
            instance = new Teacher();
        }
        return instance;
    }

    public void checkLabWork(Student student) {
        System.out.println("Преподаватель проверяет лабораторную работу
студента " + student.getName());
    }

    public void conductConsultation(Student student) {
        System.out.println("Преподаватель проводит консультацию со студентом
" + student.getName());
    }

    public void takeExam(Student student) {
        System.out.println("Преподаватель принимает экзамен у студента " +
student.getName());
    }
```

```
    public void gradeStudent(Student student) {
        System.out.println("Преподаватель выставляет отметку студенту " +
student.getName());
    }

    public void conductLecture() {
        System.out.println("Преподаватель проводит лекцию");
    }

    public void addStudent(Student student) {
        students.add(student);
    }

    public void removeStudent(Student student) {
        students.remove(student);
    }

    public List<Student> getStudents() {
        return students;
    }
}

public class Lab6_1 {
    public static void main(String[] args) {
        Teacher teacher = Teacher.getInstance();

        Student student1 = new Student("Иванов");
        Student student2 = new Student("Петров");
        Student student3 = new Student("Сидоров");

        teacher.addStudent(student1);
        teacher.addStudent(student2);
        teacher.addStudent(student3);

        for (Student student : teacher.getStudents()) {
            teacher.checkLabWork(student);
            teacher.conductConsultation(student);
            teacher.takeExam(student);
            teacher.gradeStudent(student);
        }

        teacher.conductLecture();
    }
}
```

Задание 2

ДУ автомобиля. Реализовать иерархию автомобилей для конкретных производителей и иерархию средств дистанционного управления. Автомобили должны иметь присущие им атрибуты и функции. ДУ имеет три основные функции – удаленная активация сигнализации, удаленное открытие/закрытие дверей и удаленный запуск двигателя. Эти функции должны отличаться по своим характеристикам для различных устройств ДУ.

Код программы

```
package Lab6_1;
interface RemoteControl {
    void activateAlarm();
    void toggleDoors();
    void startEngine();
}

class SimpleRemoteControl implements RemoteControl {
    public void activateAlarm() {
        System.out.println("Простое ДУ: Активация сигнализации");
    }

    public void toggleDoors() {
        System.out.println("Простое ДУ: Открытие/закрытие дверей");
    }

    public void startEngine() {
        System.out.println("Простое ДУ: Запуск двигателя");
    }
}

class AdvancedRemoteControl implements RemoteControl {
    public void activateAlarm() {
        System.out.println("Продвинутое ДУ: Активация сигнализации с
дополнительной защитой");
    }

    public void toggleDoors() {
        System.out.println("Продвинутое ДУ: Открытие/закрытие дверей с
использованием датчиков");
    }

    public void startEngine() {
        System.out.println("Продвинутое ДУ: Запуск двигателя с помощью
удаленного зажигания");
    }
}

interface Car {
    void start();
    void stop();
}

class ToyotaCar implements Car {
    public void start() {
```

```
        System.out.println("Автомобиль Toyota: Запуск двигателя");
    }

    public void stop() {
        System.out.println("Автомобиль Toyota: Остановка двигателя");
    }
}

class FordCar implements Car {
    public void start() {
        System.out.println("Автомобиль Ford: Запуск двигателя");
    }

    public void stop() {
        System.out.println("Автомобиль Ford: Остановка двигателя");
    }
}

abstract class CarRemoteControl {
    protected RemoteControl remoteControl;

    public CarRemoteControl(RemoteControl remoteControl) {
        this.remoteControl = remoteControl;
    }

    abstract void activateAlarm();
    abstract void toggleDoors();
    abstract void startEngine();
}

class ToyotaRemoteControl extends CarRemoteControl {
    private ToyotaCar car;

    public ToyotaRemoteControl(RemoteControl remoteControl, ToyotaCar car) {
        super(remoteControl);
        this.car = car;
    }

    public void activateAlarm() {
        remoteControl.activateAlarm();
    }

    public void toggleDoors() {
        remoteControl.toggleDoors();
    }

    public void startEngine() {
        car.start();
    }
}

class FordRemoteControl extends CarRemoteControl {
    private FordCar car;

    public FordRemoteControl(RemoteControl remoteControl, FordCar car) {
```

```
        super(remoteControl);
        this.car = car;
    }

    public void activateAlarm() {
        remoteControl.activateAlarm();
    }

    public void toggleDoors() {
        remoteControl.toggleDoors();
    }

    public void startEngine() {
        car.start();
    }
}

public class Lab6_2 {
    public static void main(String[] args) {
        RemoteControl simpleRemoteControl = new SimpleRemoteControl();
        RemoteControl advancedRemoteControl = new AdvancedRemoteControl();

        ToyotaCar toyotaCar = new ToyotaCar();
        FordCar fordCar = new FordCar();

        CarRemoteControl toyotaRemoteControl = new
ToyotaRemoteControl(simpleRemoteControl, toyotaCar);
        CarRemoteControl fordRemoteControl = new
FordRemoteControl(advancedRemoteControl, fordCar);

        System.out.println("Toyota Car:");
        toyotaRemoteControl.activateAlarm();
        toyotaRemoteControl.toggleDoors();
        toyotaRemoteControl.startEngine();

        System.out.println("\nFord Car:");
        fordRemoteControl.activateAlarm();
        fordRemoteControl.toggleDoors();
        fordRemoteControl.startEngine();
    }
}
```

Задание 3

Проект «Пиццерия». Реализовать формирование заказ(а)ов, их отмену, а также повторный заказ с теми же самыми позициями.

Код программы

```
package Lab6_1;
import java.util.ArrayList;
import java.util.List;

class Pizza {
    private String name;

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

    public String getName() {
        return name;
    }
}

interface Command {
    void execute();
}

class AddPizzaCommand implements Command {
    private Pizza pizza;
    private List<Pizza> order;

    public AddPizzaCommand(Pizza pizza, List<Pizza> order) {
        this.pizza = pizza;
        this.order = order;
    }

    public void execute() {
        order.add(pizza);
        System.out.println(pizza.getName() + " Добавлен в заказ.");
    }
}

class CancelOrderCommand implements Command {
    private List<Pizza> order;

    public CancelOrderCommand(List<Pizza> order) {
        this.order = order;
    }

    public void execute() {
        order.clear();
        System.out.println("Заказ отменен.");
    }
}

class RepeatOrderCommand implements Command {
```

```
    private List<Pizza> order;
    private List<Pizza> previousOrder;

    public RepeatOrderCommand(List<Pizza> order, List<Pizza> previousOrder) {
        this.order = order;
        this.previousOrder = previousOrder;
    }

    public void execute() {
        order.addAll(previousOrder);
        System.out.println("Заказ продублирован.");
    }
}

class Pizzeria {
    private List<Pizza> order;
    private List<Pizza> previousOrder;
    private List<Command> commands;

    public Pizzeria() {
        order = new ArrayList<>();
        previousOrder = new ArrayList<>();
        commands = new ArrayList<>();
    }

    public void addPizzaToOrder(Pizza pizza) {
        commands.add(new AddPizzaCommand(pizza, order));
    }

    public void cancelOrder() {
        commands.add(new CancelOrderCommand(order));
    }

    public void repeatOrder() {
        commands.add(new RepeatOrderCommand(order, previousOrder));
    }

    public void executeCommands() {
        for (Command command : commands) {
            command.execute();
        }
        previousOrder.clear();
        previousOrder.addAll(order);
        commands.clear();
    }
}

public class Lab6_3 {
    public static void main(String[] args) {
        Pizzeria pizzeria = new Pizzeria();
        Pizza pizza1 = new Pizza("Маргарита");
        Pizza pizza2 = new Pizza("Пепперони");

        pizzeria.addPizzaToOrder(pizza1);
        pizzeria.addPizzaToOrder(pizza2);
```

```
        pizzeria.executeCommands();

        pizzeria.cancelOrder();
        pizzeria.executeCommands();

        pizzeria.repeatOrder();
        pizzeria.executeCommands();
    }
}
```