

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

**Лабораторная работа №6**

**По дисциплине:** «Современные платформы программирования»

**Тема:** «Паттернов проектирования в Java»

**Выполнил:**

Студент 3 курса

Группы ПО-8

Бувин Д.А.

**Проверил:**

А. А. Крощенко

**Брест, 2024**

## Лабораторная работа №6

### Вариант 7

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

#### **Задание №1:**

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

Для реализации задания можно использовать паттерн "Наблюдатель" (Observer).

Паттерн "Наблюдатель" позволяет создать механизм подписки и уведомления, где объекты-наблюдатели (студенты) могут подписываться на изменения в объекте-издатель (преподаватель), и когда происходят эти изменения, все подписанные объекты автоматически уведомляются.

#### **Код программы:**

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

interface Observer {
    void update(String message);
}

interface Subject {
    void addObserver(Observer observer);
    void removeObserver(Observer observer);
    void notifyObservers(String message);
}

class Teacher implements Subject {
    private List<Observer> students;

    public Teacher() {
        this.students = new ArrayList<>();
    }
}
```

```

    }

    @Override
    public void addObserver(Observer observer) {
        students.add(observer);
    }

    @Override
    public void removeObserver(Observer observer) {
        students.remove(observer);
    }

    @Override
    public void notifyObservers(String message) {
        for (Observer student : students) {
            student.update(message);
        }
    }

    public void checkLabWork() {
        String message = "Лабораторная работа проверена";
        notifyObservers(message);
    }
}

class Student implements Observer {
    private String name;

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

    @Override
    public void update(String message) {
        System.out.println("Студент " + name + " получил уведомление: " + message);
    }
}

public class Task_1 {
    public static void main(String[] args) {
        Teacher teacher = new Teacher();

        Student student1 = new Student("Иван");
        Student student2 = new Student("Петр");
    }
}

```

```
teacher.addObserver(student1);
teacher.addObserver(student2);

teacher.checkLabWork();
}
}
```

### Результат работы:

```
Студент Иван получил уведомление: Лабораторная работа проверена
Студент Петр получил уведомление: Лабораторная работа проверена

Process finished with exit code 0
```

### 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.

Для реализации данной задачи подходит паттерн "Абстрактная фабрика" (Abstract Factory).

Паттерн "Абстрактная фабрика" позволяет создавать семейства взаимосвязанных объектов без указания их конкретных классов. Он хорошо подходит для создания иерархии объектов различных типов (например, различные модели автомобилей разных производителей и их средства дистанционного управления), при этом обеспечивая совместимость между объектами одного семейства.

### Код программы:

```
interface CarFactory {
    Car createCar();
    RemoteControl createRemoteControl();
}
```

```
interface Car {  
    void start();  
    void stop();  
    void lockDoors();  
    void unlockDoors();  
    void activateAlarm();  
}
```

```
interface RemoteControl {  
    void activateAlarm();  
    void unlockDoors();  
    void startEngine();  
}
```

```
class BMWFactory implements CarFactory {  
    @Override  
    public Car createCar() {  
        return new BMWCar();  
    }  
  
    @Override  
    public RemoteControl createRemoteControl() {  
        return new BMWRemoteControl();  
    }  
}
```

```
class AudiFactory implements CarFactory {  
    @Override  
    public Car createCar() {  
        return new AudiCar();  
    }  
  
    @Override  
    public RemoteControl createRemoteControl() {  
        return new AudiRemoteControl();  
    }  
}
```

```
class BMWCar implements Car {  
    @Override  
    public void start() {  
        System.out.println("BMW: Двигатель запущен");  
    }  
}
```

```

@Override
public void stop() {
    System.out.println("BMW: Двигатель остановлен");
}

@Override
public void lockDoors() {
    System.out.println("BMW: Двери закрыты");
}

@Override
public void unlockDoors() {
    System.out.println("BMW: Двери разблокированы");
}

@Override
public void activateAlarm() {
    System.out.println("BMW: Сигнализация активирована");
}
}

class AudiCar implements Car {
    @Override
    public void start() {
        System.out.println("Audi: Двигатель запущен");
    }

    @Override
    public void stop() {
        System.out.println("Audi: Двигатель остановлен");
    }

    @Override
    public void lockDoors() {
        System.out.println("Audi: Двери закрыты");
    }

    @Override
    public void unlockDoors() {
        System.out.println("Audi: Двери разблокированы");
    }

    @Override
    public void activateAlarm() {

```

```

        System.out.println("Audi: Сигнализация активирована");
    }
}

class BMWRemoteControl implements RemoteControl {
    @Override
    public void activateAlarm() {
        System.out.println("BMW Remote Control: Активация сигнализации");
    }

    @Override
    public void unlockDoors() {
        System.out.println("BMW Remote Control: Разблокировка дверей");
    }

    @Override
    public void startEngine() {
        System.out.println("BMW Remote Control: Запуск двигателя");
    }
}

class AudiRemoteControl implements RemoteControl {
    @Override
    public void activateAlarm() {
        System.out.println("Audi Remote Control: Активация сигнализации");
    }

    @Override
    public void unlockDoors() {
        System.out.println("Audi Remote Control: Разблокировка дверей");
    }

    @Override
    public void startEngine() {
        System.out.println("Audi Remote Control: Запуск двигателя");
    }
}

public class Task_2 {
    public static void main(String[] args) {
        CarFactory bmwFactory = new BMWFactory();
        Car bmwCar = bmwFactory.createCar();
        RemoteControl bmwRemote = bmwFactory.createRemoteControl();
    }
}

```

```

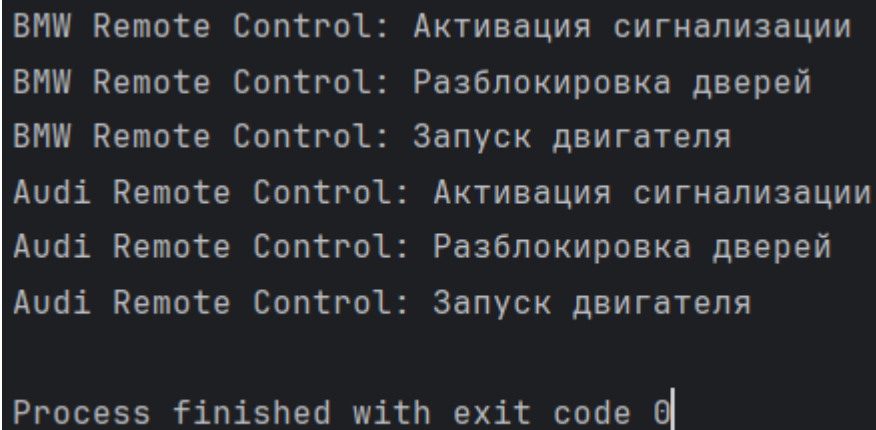
CarFactory audiFactory = new AudiFactory();
Car audiCar = audiFactory.createCar();
RemoteControl audiRemote = audiFactory.createRemoteControl();

bmwRemote.activateAlarm();
bmwRemote.unlockDoors();
bmwRemote.startEngine();

audiRemote.activateAlarm();
audiRemote.unlockDoors();
audiRemote.startEngine();
}
}

```

### Результат работы:



```

BMW Remote Control: Активация сигнализации
BMW Remote Control: Разблокировка дверей
BMW Remote Control: Запуск двигателя
Audi Remote Control: Активация сигнализации
Audi Remote Control: Разблокировка дверей
Audi Remote Control: Запуск двигателя

Process finished with exit code 0

```

### Задание №3:

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

Для реализации проекта "Пиццерия" подходит паттерн проектирования "Команда" (Command).

Паттерн "Команда" позволяет инкапсулировать запрос на выполнение определенного действия в виде объекта, позволяя клиенту параметризовать клиентские объекты с операциями, определять очередь запросов и поддерживать отмену операций.

### Код программы:

```

import java.util.ArrayList;

```



```
import java.util.List;
interface Command {
    void execute();
    void undo();
}

class AddPizzaCommand implements Command {
    private Pizza pizza;
    private Order order;

    public AddPizzaCommand(Pizza pizza, Order order) {
        this.pizza = pizza;
        this.order = order;
    }

    @Override
    public void execute() {
        order.addPizza(pizza);
    }

    @Override
    public void undo() {
        order.removePizza(pizza);
    }
}

class CancelOrderCommand implements Command {
    private Order order;

    public CancelOrderCommand(Order order) {
        this.order = order;
    }

    @Override
    public void execute() {
        order.cancel();
    }

    @Override
    public void undo() {
        order.restore();
    }
}
```

```

class Order {
    private List<Pizza> pizzas = new ArrayList<>();
    private boolean canceled = false;

    public void addPizza(Pizza pizza) {
        pizzas.add(pizza);
    }

    public void removePizza(Pizza pizza) {
        pizzas.remove(pizza);
    }

    public void cancel() {
        pizzas.clear();
        canceled = true;
    }

    public void restore() {
        canceled = false;
    }

    public void print() {
        if (canceled) {
            System.out.println("Заказ отменен");
        } else {
            System.out.println("Заказ:");
            for (Pizza pizza : pizzas) {
                System.out.println("- " + pizza);
            }
        }
    }
}

```

```

class Pizza {
    private String name;

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

    @Override
    public String toString() {
        return name;
    }
}

```

```

}

public class Task_3 {
    public static void main(String[] args) {
        Order order = new Order();
        Pizza pizza1 = new Pizza("Пепперони");
        Pizza pizza2 = new Pizza("Маргарита");

        Command addPizza1Command = new AddPizzaCommand(pizza1, order);
        Command addPizza2Command = new AddPizzaCommand(pizza2, order);

        addPizza1Command.execute();
        addPizza2Command.execute();

        order.print();

        Command cancelOrderCommand = new CancelOrderCommand(order);
        cancelOrderCommand.execute();

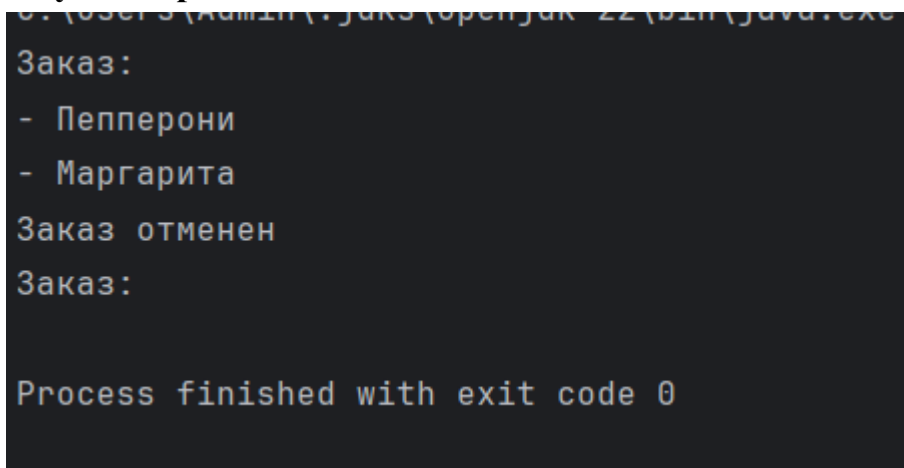
        order.print();

        cancelOrderCommand.undo();

        order.print();
    }
}

```

### Результат работы:



```

C:\Users\kashin\IdeaProjects\Task_22\bin\java.exe
Заказ:
- Пепперони
- Маргарита
Заказ отменен
Заказ:

Process finished with exit code 0

```

**Вывод:** По итогу выполнения лабораторной работы, я приобрел практические навыки применения паттернов проектирования при решении практических задач с использованием языка Java.