

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

ОТЧЁТ
по лабораторной работе №6

Выполнила:
студентка группы ПО-9
Кот А. А.

Проверил:
Крощенко А. А.

Брест 2024

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

Вариант 7.

Ход работы

Прочитать задания, взятые из каждой группы.

- Определить паттерн проектирования, который может использоваться при реализации задания. Пояснить свой выбор.

- Реализовать фрагмент программной системы, используя выбранный паттерн.

Реализовать все необходимые дополнительные классы.

Варианты работ определяются по последней цифре в зачетной книжке.

Задание 1.

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

Для реализации задания был выбран паттерн Наблюдатель. В этом случае класс Преподаватель будет выступать в роли наблюдаемого объекта, а объекты класса Студент будут выступать в роли наблюдателей. Когда преподаватель проводит какое-либо мероприятие (например, проверяет лабораторную работу, проводит консультацию, принимает экзамен и т. д.), он оповещает всех своих наблюдателей (студентов) о произошедшем событии. Студенты, подписавшись на преподавателя, получают уведомления о событиях и могут реагировать соответственно. Таким образом, паттерн "Наблюдатель" позволяет реализовать одновременное взаимодействие преподавателя с несколькими студентами, обеспечивая эффективную коммуникацию между ними.

Работа программы:

```
The teacher has interacted with the student. The following event occurred: takeExam
The teacher has interacted with the student. The following event occurred: holdConsultation
The teacher has interacted with the student. The following event occurred: holdLecture
The teacher has interacted with the student. The following event occurred: reviewLabwork
The teacher has interacted with the student. The following event occurred: reviewLabwork
The teacher has interacted with the student. The following event occurred: putGrade
The teacher has interacted with the student. The following event occurred: putGrade

Process finished with exit code 0
```

Код программы:

Task1.java

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

class TeacherEventNotifier {
    Map<String, List<TeacherEventListener>> students = new HashMap<>();

    public TeacherEventNotifier(String... events) {
```

```

        for (String event : events) {
            this.students.put(event, new ArrayList<>());
        }
    }

    public void subscribe (String event, TeacherEventListener student) {
        List<TeacherEventListener> subscribers = students.get(event);
        subscribers.add(student);
    }

    public void unsubscribe (String event, TeacherEventListener student) {
        List<TeacherEventListener> subscribers = students.get(event);
        subscribers.remove(student);
    }

    public void notify (String event) {
        List<TeacherEventListener> subscribers = students.get(event);
        for (TeacherEventListener student : subscribers) {
            student.update(event);
        }
    }
}

class Teacher {
    public TeacherEventNotifier events;

    public Teacher () {
        this.events = new TeacherEventNotifier("reviewLabwork",
"holdConsultation",
        "takeExam", "putGrade", "holdLecture");
    }

    void reviewLabwork () {
        events.notify("reviewLabwork");
    }

    void holdConsultation () {
        events.notify("holdConsultation");
    }

    void takeExam () {
        events.notify("takeExam");
    }

    void putGrade () {
        events.notify("putGrade");
    }

    void holdLecture () {
        events.notify("holdLecture");
    }
}

interface TeacherEventListener {
    void update(String eventType);
}

class Student implements TeacherEventListener {
    @Override
    public void update(String eventType) {
        System.out.println("The teacher has interacted with the student. " +
            "The following event occurred: " + eventType);
    }
}

public class Task1 {

```

```

public static void main(String[] args) {
    Teacher teacher = new Teacher();
    teacher.events.subscribe("reviewLabwork", new Student());
    teacher.events.subscribe("reviewLabwork", new Student());
    teacher.events.subscribe("holdConsultation", new Student());
    teacher.events.subscribe("takeExam", new Student());
    teacher.events.subscribe("putGrade", new Student());
    teacher.events.subscribe("putGrade", new Student());
    teacher.events.subscribe("holdLecture", new Student());

    teacher.takeExam();
    teacher.holdConsultation();
    teacher.holdLecture();
    teacher.reviewLabwork();
    teacher.putGrade();
}
}

```

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 выполнения данного задания был выбран паттерн Мост. Класс RemoteControl и интерфейс Car будут выступать в качестве моста, если впоследствии будет требоваться расширение программы для добавления разных вариаций пультов, это можно будет легко сделать за счет наследования от класса RemoteControl, который напрямую взаимодействует с интерфейсом Машин.

Работа программы:

```

Ford Client:
Ford is honking at ya!
Your Ford door's configuration has been changed!
Ford has started: vroom-vroom
-----
Toyota Client:
Toyota ain't honking at you. Only staring.
Toyota has changed door status.
Toyota is ready to roll.

Process finished with exit code 0

```

Код программы:

Task2.java

```

interface Car {
    void honk();
    void doors();
    void start();
}

class Ford implements Car {
    @Override
    public void honk() {
        System.out.println("Ford is honking at ya!");
    }
}

```

```

    }

    @Override
    public void doors() {
        System.out.println("Your Ford door's configuration has been
changed!");
    }

    @Override
    public void start() {
        System.out.println("Ford has started: vroom-vroom");
    }
}

class Toyota implements Car {
    @Override
    public void honk() {
        System.out.println("Toyota ain't honking at you. Only staring.");
    }

    @Override
    public void doors() {
        System.out.println("Toyota has changed door status.");
    }

    @Override
    public void start() {
        System.out.println("Toyota is ready to roll.");
    }
}

class CarRemote {
    private final Car car;

    CarRemote(Car car) {
        this.car = car;
    }

    public void activateSignaling() {
        car.honk();
    }

    public void manipulateDoors() {
        car.doors();
    }

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

public class Task2 {
    public static void main(String[] args) {
        Car ford = new Ford();
        CarRemote fordRemote = new CarRemote(ford);

        System.out.println("Ford Client:");
        fordRemote.activateSignaling();
        fordRemote.manipulateDoors();
        fordRemote.startEngine();

        System.out.println("-----");

        Car toyota = new Toyota();
        CarRemote toyotaRemote = new CarRemote(toyota);
    }
}

```

```

        System.out.println("Toyota Client:");
        toyotaRemote.activateSignaling();
        toyotaRemote.manipulateDoors();
        toyotaRemote.startEngine();
    }
}

```

Задание 3.

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

Для выполнения задания был выбран паттерн Сообщение. Паттерн "Команда" позволяет инкапсулировать запрос как объект, позволяя параметризовать клиентов с запросами, оформлять запросы в виде объектов, ставить запросы в очередь, а также поддерживать отмену операций. При использовании паттерна "Команда" у нас будет гибкая система управления заказами, которая позволит легко добавлять новые функции-команды и расширять поведение без изменения основного кода.

Работа программы:

```

Заказ сформирован: Заказ: [Пепперони - $10.99, Маргарита - $5.99]
Повторный заказ сформирован: Заказ: [Пепперони - $10.99, Маргарита - $5.99]
Заказ отменен: Заказ: []

Process finished with exit code 0

```

Код программы:

Task3.java

```

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

interface Command {
    void execute();
}

class CreateOrderCommand implements Command {
    private Order order;
    private List<MenuItem> itemsToAdd;

    public CreateOrderCommand(Order order, List<MenuItem> itemsToAdd) {
        this.order = order;
        this.itemsToAdd = itemsToAdd;
    }

    @Override
    public void execute() {
        order.addItem(itemsToAdd);
        System.out.println("Заказ сформирован: " + order);
    }
}

class CancelOrderCommand implements Command {
    private Order order;

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

    @Override

```

```

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

    class RepeatOrderCommand implements Command {
        private OrderHistory orderHistory;
        private int orderIndex;

        public RepeatOrderCommand(OrderHistory orderHistory, int orderIndex) {
            this.orderHistory = orderHistory;
            this.orderIndex = orderIndex;
        }

        @Override
        public void execute() {
            Order orderToRepeat = orderHistory.getOrder(orderIndex);
            Order repeatedOrder = new Order();
            repeatedOrder.addItem(orderToRepeat.getItems());
            System.out.println("Повторный заказ сформирован: " + repeatedOrder);
        }
    }

    class Order {
        private List<MenuItem> items;

        public Order() {
            this.items = new ArrayList<>();
        }

        public void addItem(List<MenuItem> itemsToAdd) {
            items.addAll(itemsToAdd);
        }

        public void cancel() {
            items.clear();
        }

        public List<MenuItem> getItems() {
            return items;
        }

        @Override
        public String toString() {
            return "Заказ: " + items;
        }
    }

    class OrderHistory {
        private List<Order> orders;

        public OrderHistory() {
            this.orders = new ArrayList<>();
        }

        public void addOrder(Order order) {
            orders.add(order);
        }

        public Order getOrder(int index) {
            return orders.get(index);
        }
    }

    class MenuItem {

```

```

private String name;
private double price;

public MenuItem(String name, double price) {
    this.name = name;
    this.price = price;
}

@Override
public String toString() {
    return name + " - $" + price;
}
}

class Waiter {
    private List<Command> orders;

    public Waiter() {
        this.orders = new ArrayList<>();
    }

    public void takeOrder(Command command) {
        orders.add(command);
    }

    public void placeOrders() {
        for (Command command : orders) {
            command.execute();
        }
        orders.clear();
    }
}

public class Task3 {
    public static void main(String[] args) {
        MenuItem pepperoni = new MenuItem("Пепперони", 10.99);
        MenuItem margarita = new MenuItem("Маргарита", 5.99);

        Order order = new Order();

        Waiter waiter = new Waiter();

        List<MenuItem> itemsToAdd = new ArrayList<>();
        itemsToAdd.add(pepperoni);
        itemsToAdd.add(margarita);
        Command createOrderCommand = new CreateOrderCommand(order,
itemsToAdd);

        Command cancelOrderCommand = new CancelOrderCommand(order);

        OrderHistory orderHistory = new OrderHistory();
        orderHistory.addOrder(order);
        Command repeatOrderCommand = new RepeatOrderCommand(orderHistory, 0);

        waiter.takeOrder(createOrderCommand);
        waiter.takeOrder(repeatOrderCommand);
        waiter.takeOrder(cancelOrderCommand);

        waiter.placeOrders();
    }
}

```

Вывод: в ходе лабораторной работы мы приобрели навыки применения паттернов проектирования при решении практических задач с использованием языка Java.