Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6

По дисциплине: «Современные платформы программирования»

Выполнил:
студент 3 курса
группы ПО-8
Сорока В.С.

Проверил:
Крощенко А.А.

Брест, 2024

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

- Определить паттерн проектирования, который может использоваться при реализации задания. Пояснить свой выбор.
- Реализовать фрагмент программной системы, используя выбранный паттерн. Реализовать все необходимые дополнительные классы.

Вариант 6

Задание 1

Музыкальный магазин. Должно обеспечиваться одновременное обслуживание нескольких покупателей. Магазин должен предоставлять широкий выбор товаров различных музыкальных направлений.

Выбранный паттерн проектирования: Стратегия (Strategy)

Паттерн идеально подходит для реализации магазина, где несколько покупателей могут одновременно совершать покупки, а ассортимент охватывает различные музыкальные направления.

Спецификации ввода-вывода программы:

1. Интерфейс MusicStoreStrategy: определяет методы для обработки заказа.
2. Реализации стратегии:
   А) StandardStrategy: обрабатывает стандартный заказ без скидок.
   Б) DiscountStrategy: обрабатывает заказ с применением скидки, если сумма заказа превышает DISCOUNT_THRESHOLD или количество товаров больше MINIMUM_PRODUCTS_FOR_DISCOUNT.
   В) VIPStrategy: Обрабатывает VIP-заказ без скидок.

Класс MusicStore:
- Содержит ссылку на текущую стратегию обработки заказа.
- Предоставляет методы processOrder для обработки заказа с использованием текущей стратегии и setStrategy для изменения стратегии.

Классы Customer и Order:
- Хранят информацию о покупателе и заказе соответственно.

Класс Product:
- Хранит информацию о товаре, включая его название, цену и оригинальную цену.
- Предоставляет метод applyDiscount для применения скидки к цене товара.

Класс MusicStoreApp:
- Создает экземпляры MusicStore с различными стратегиями.

- Создает покупателей и заказы.
- Демонстрирует обработку заказов с использованием разных стратегий.

3. Входные данные
   - Список товаров для каждого заказа.
   - Тип стратегии обработки заказа для каждого заказа.

4. Выходные данные
   - Информация о каждом обрабатываемом заказе:
   - Имя покупателя.
   - Список товаров.
   - Примененная скидка (если есть).
   - Общая сумма заказа.

Текст программы

```
// Интерфейс Strategy
interface MusicStoreStrategy
{
    void processOrder(Customer customer, Order order);
}

// Реализация стратегий
class StandardStrategy implements MusicStoreStrategy
{
    @Override
    public void processOrder(Customer customer, Order order)
    {
        // Обработка стандартного заказа
        System.out.println("\nОбрабатывается стандартный заказ для " +
customer.getName() + ".");
        System.out.println("Товары:");
        for (Product product : order.getProducts()) {
            System.out.println("- " + product.getName() + " (" +
String.format("%.0f p.", product.getPrice()) + ")");
        }
        order.displayTotalPrice();
    }
}

class DiscountStrategy implements MusicStoreStrategy
{
    // Константы для скидок
    private static final double DISCOUNT_THRESHOLD = 1000.0; // Порог суммы
заказа для скидки
    private static final double DISCOUNT_RATE = 0.1; // Ставка скидки (10%)
    private static final int MINIMUM_PRODUCTS_FOR_DISCOUNT = 3; //
Минимальное количество товаров для скидки

    @Override
    public void processOrder(Customer customer, Order order)
    {
        // Обработка заказа с применением скидок
        System.out.println("\nОбрабатывается заказ с применением скидок для "
+ customer.getName() + ".");
        System.out.println("Товары:");
        for (Product product : order.getProducts()) {
            System.out.println("- " + product.getName() + " (" +
String.format("%.0f p.", product.getPrice()) + ")");
        }
```

```
        // Проверяем сумму заказа для возможной скидки
        double totalPrice = order.getTotalPrice();
        if (totalPrice >= DISCOUNT_THRESHOLD || order.getProducts().size() >=
MINIMUM_PRODUCTS_FOR_DISCOUNT) {
            applyDiscount(order);
        }
        // Выводим общую сумму
        order.displayTotalPrice();
    }

    // Применение скидки
    private void applyDiscount(Order order)
    {
        System.out.println("\nПрименяется скидка в размере " + DISCOUNT_RATE
* 100 + "%");
        List<Product> products = order.getProducts();
        double totalDiscount = 0.0; // Инициализируем переменную для хранения
общей суммы скидки
        for (Product product : products) {
            double discountAmount = product.getOriginalPrice() *
DISCOUNT_RATE;
            double discountedPrice = product.getOriginalPrice() -
discountAmount;
            product.applyDiscount(discountedPrice);
            totalDiscount += discountAmount; // Обновляем общую сумму скидки
        }
        System.out.println("Общая скидка: " + totalDiscount + " руб."); //
Выводим общую сумму скидки
    }
}


class VIPStrategy implements MusicStoreStrategy
{
    @Override
    public void processOrder(Customer customer, Order order)
    {
        // Обработка VIP-заказа
        System.out.println("\nОбрабатывается VIP-заказ для " +
customer.getName() + ".");
        System.out.println("Товары:");
        for (Product product : order.getProducts()) {
            System.out.println("- " + product.getName() + " (" +
String.format("%.0f p.", product.getPrice()) + ")");
        }
        order.displayTotalPrice();
    }
}

// Класс MusicStore
class MusicStore
{
    private MusicStoreStrategy strategy;

    public MusicStore(MusicStoreStrategy strategy) {
        this.strategy = strategy;
    }

    public void processOrder(Customer customer, Order order) {
        strategy.processOrder(customer, order);
    }

    public void setStrategy(MusicStoreStrategy strategy) {
        this.strategy = strategy;
    }
}
```

```
public class MusicStoreApp
{
    public static void main(String[] args)
    {
        // Создание магазина с базовой стратегией
        MusicStore store = new MusicStore(new StandardStrategy());
        // Создание магазина с discount стратегией
        MusicStore storeDiscount = new MusicStore(new DiscountStrategy());

        // Создание покупателей и заказов
        Customer customer1 = new Customer("Иванов Иван");
        Order order1 = new Order(customer1, Arrays.asList(new Product("Гитара
пятиструночная", 1000), new Product("Струны гитарные", 50)));
        Customer customer2 = new Customer("Сергеев Сергей");
        Order order2 = new Order(customer2, Arrays.asList(new
Product("Виолончель", 800), new Product("Балалайка", 400)));
        store.processOrder(customer1, order1);
        store.processOrder(customer2, order2);

        store.setStrategy(new VIPStrategy());
        // VIP-заказ
        Customer vipCustomer = new Customer("Алексеев Алексей");
        Order vipOrder = new Order(vipCustomer, Arrays.asList(new
Product("Пианино Леруаль", 5000), new Product("Гитара ДиКалино", 25000)));
        store.processOrder(vipCustomer, vipOrder);

        // Переключение на Discount-стратегию
        store.setStrategy(new DiscountStrategy());
        // Создание покупателей и заказов
        Customer customer3 = new Customer("Мариевич Мария");
        Order order3 = new Order(customer3, Arrays.asList(new Product("Гитара
четырёхструночная", 800), new Product("Струны гитарные", 70), new
Product("Набор инструментов для настройки гитары", 250)));
        store.processOrder(customer3, order3);
    }
}
```

Пример работы программы

```
Обрабатывается стандартный заказ для Иванов Иван.
Товары:
- Гитара пятиструночная (1000 р.)
- Струны гитарные (50 р.)
Общая сумма: 1050 р.

Обрабатывается стандартный заказ для Сергеев Сергей.
Товары:
- Виолончель (800 р.)
- Балалайка (400 р.)
Общая сумма: 1200 р.

Обрабатывается VIP-заказ для Алексеев Алексей.
Товары:
- Пианино Леруаль (5000 р.)
- Гитара ДиКалино (25000 р.)
Общая сумма: 30000 р.

Обрабатывается заказ с применением скидок для Мариевич Мария.
Товары:
- Гитара четырёхструночная (800 р.)
- Струны гитарные (70 р.)
- Набор инструментов для настройки гитары (250 р.)

Применяется скидка в размере 10.0%
- Скидка на: Гитара четырёхструночная = 80.0 руб.
- Скидка на: Струны гитарные = 7.0 руб.
- Скидка на: Набор инструментов для настройки гитары = 25.0 руб.
Общая скидка: 112.0 руб.
Общая сумма: 1008 р.
```

Задание 2

Учетная запись покупателя книжного интернет-магазина. Предусмотреть различные уровни учётки в зависимости от активности покупателя. Дополнительные уровни добавляют функциональные возможности и открывают доступ к уникальным предложениям.

Выбранный паттерн: Декоратор

Позволяют динамически добавлять объектам новую функциональность, оборачивая их в разные обёртки

Спецификации ввода-вывода программы:

Клиенты имеют уровни, которые влияют на доступные им функции. Система использует паттерн Декоратор для динамического добавления функциональности к базовому уровню клиента.

1. Классы
   А) Client:
   - Представляет клиента системы.
   - Имеет ссылку на текущий уровень клиента.
   - Предоставляет методы для изменения уровня клиента (toSecondLevel, toThirdLevel), получения текущего уровня (getLevel), покупки книги (buyBook), добавления книги в избранное (addToFavourites) и оставления оценки книге (leaveMark).

   Б) Level:
   - Интерфейс, определяющий методы для всех уровней клиента (getLevel, buyBook, addToFavourites, leaveMark).

   В) Decorator:
   - Абстрактный класс-декоратор, наследующий от интерфейса Level.
   - Хранит ссылку на декорируемый уровень (mLevel).
   - Реализует базовые методы интерфейса Level, делегируя вызовы соответствующему уровню (mLevel).

   Г) FirstLevel, SecondLevel, ThirdLevel:
   - Конкретные реализации уровней клиента, наследующие от Decorator.
   - Переопределяют методы addToFavourites и leaveMark в зависимости от возможностей уровня.
   - FirstLevel - базовый уровень с ограниченной функциональностью.
   - SecondLevel - предоставляет возможность добавлять книги в избранное.
   - ThirdLevel - расширяет SecondLevel возможностью оставлять оценки книгам и покупать книги.

3. Входные данные
   - Выбор действия клиента (getLevel, addToFavourites, leaveMark, buyBook).
   - Название книги (для addToFavourites, leaveMark, buyBook).

- Оценка книги (для leaveMark).

4. Выходные данные
   - Текущий уровень клиента (getLevel).
   - Сообщение об успехе или ошибке выполнения действия (addToFavourites, leaveMark, buyBook).

Текст программы

```
class Decorator implements Level
{

    protected Level mLevel;
    protected String mLevelName;

    public Decorator(Level level) {
        this.mLevel = level;
    }

    @Override
    public String getLevel() {
        return this.mLevel.getLevel();
    }
    @Override
    public String buyBook(String name) {
        return this.mLevel.buyBook(name);
    }
    @Override
    public String addToFavourites(String name) {
        return this.mLevel.addToFavourites(name);
    }
    @Override
    public String leaveMark(String name, String mark) {
        return this.mLevel.leaveMark(name, mark);
    }
}
```

```
interface Level
{
    public String getLevel();
    public String buyBook(String name);
    public String addToFavourites(String name);
    public String leaveMark(String name, String mark);
}
```

```
class FirstLevel implements Level
{
    private String mLevelName;

    public FirstLevel() {
        this.mLevelName = "[1]";
    }

    @Override
    public String getLevel() {
        return this.mLevelName;
    }
    @Override
    public String buyBook(String name) {
        return name;
    }
    @Override
    public String addToFavourites(String name) {
        return "Вы не можете добавить в Избранное на уровне [1]    :(";
    }
```

```
    @Override
    public String leaveMark(String name, String mark) {
        return "Вы не можете поставить оценку на уровне [1]        :(";
    }
}
```

```
class SecondLevel extends Decorator
{
    public SecondLevel(Level level) {
        super(level);
        this.mLevelName = "[2] -> " + super.getLevel();
    }

    @Override
    public String getLevel() {
        return this.mLevelName;
    }

    @Override
    public String buyBook(String name) {
        return this.mLevel.buyBook(name);
    }

    @Override
    public String addToFavourites(String name) {
        return "Вы добавили книгу [ " + name + " ] в Избранное";
    }

    @Override
    public String leaveMark(String name, String mark) {
        return "Вы не можете поставить оценку на уровне [2]        :(";
    }
}
```

Пример работы программы:

```
[1]
Вы не можете добавить в Избранное на уровне [1]    :(
Вы не можете поставить оценку на уровне [1]        :(

[2] -> [1]
Вы добавили книгу [ Иван Дурак и серый волк ] в Избранное
Вы не можете поставить оценку на уровне [2]        :(

[3] -> [2] -> [1]
Вы добавили книгу [ Благородство из степи ] в Избранное
Вы поставили оценку [7] для книги [ Война и мир. Том 3 ]
Вы добавили книгу [ Большая энциклопедия животных ] в корзину
```

Задание 3

Проект «Принтер». Предусмотреть выполнение операций (печать, загрузка бумаги, извлечение зажатой бумаги, заправка картриджа), режимы – ожидание, печать документа, зажатие бумаги, отказ – при отсутствии бумаги или краски, атрибуты – модель, количество листов в лотке, % краски в картридже, вероятность зажатия.

Выбранный паттерн: Состояние
В зависимости от состояния объект принтер изменяет поведение.

Спецификация ввода-вывода
1. Описание
Класс Printer:
- Моделирует принтер с его характеристиками (модель, количество бумаги, объем краски, вероятность замятия бумаги).
- Содержит ссылку на текущее состояние принтера.
- Предоставляет методы для печати, загрузки бумаги, извлечения замятой бумаги и заправки картриджа.

Интерфейс IPrinter:
- Определяет методы для взаимодействия с принтером.

Абстрактный класс State:
- Определяет базовые методы для каждого состояния принтера.

Классы State:
- WaitState: состояние ожидания печати.
- PrintState: состояние печати.
- ClampingState: состояние замятия бумаги.
- FailureState: состояние сбоя (недостаточно бумаги или краски).

Класс Random:
- Используется для генерации случайных чисел для имитации вероятности замятия бумаги.

2. Входные данные
   - Количество бумаги, которое пользователь хочет распечатать.
   - Действия пользователя (печать, загрузка бумаги, извлечение замятой бумаги, заправка картриджа).

3. Выходные данные
   - Сообщения о текущем состоянии принтера и его действиях.
   - Информация о количестве бумаги и краски после каждого действия.
   - Сообщения о замятии бумаги, недостатке бумаги или краски.

Код программы

```
abstract class State
{
    Printer mPrinter;

    State(Printer printer) {
        this.mPrinter = printer;
    }

    public abstract String onPrint(int numPaper);
    public abstract String onLoadPaper(int numPaper);
    public abstract String onExtractClampingPaper();
    public abstract String onRefilleCartridge();
}
```

```
class WaitState extends State
{
    WaitState(Printer printer) {
        super(printer);
    }

    @Override
    public String onPrint(int numPaper) {
        this.mPrinter.changeState(new PrintState(this.mPrinter));
        return "Состояние ожидания --> Этап печати.\n";
    }
    @Override
    public String onLoadPaper(int numPaper) {
        return "Бумага загружена.\nСостояние ожидания.\n";
    }
    @Override
    public String onExtractClampingPaper() {
        return "Бумага извлечена.\nСостояние ожидания.\n";
    }
    @Override
    public String onRefilleCartridge() {
        this.mPrinter.setPaintVolume(100);
        return "Картридж был заправлен.\nСостояние ожидания.\n";
    }
}
```

```
class Printer implements IPrinter
{
    private String mModel;
    private int mNumPapers;
    private double mPaintVolume;
    private double mPinchingProbability;
    private State mState;

    public Printer(String model, int numPapers, double paintVolume, double
pinchingProbability)
    {
        this.setModel(model);
        this.setNumPapers(numPapers);
        this.setPaintVolume(paintVolume);
        this.setPinchingProbability(pinchingProbability);
        this.changeState(new WaitState(this));
    }

    @Override
    public void changeState(State state) {
        this.mState = state;
    }

    public State getSatate() {
        return this.mState;
    }
```

```
    public void setModel(String model) {
        this.mModel = model;
    }

    public String getModel() {
        return this.mModel;
    }

    public void setNumPapers(int numPapers) {
        this.mNumPapers = numPapers;
    }

    public int getNumPapers() {
        return this.mNumPapers;
    }

    public void setPaintVolume(double paintVolume) {
        this.mPaintVolume = paintVolume;
    }

    public double getPaintVolume() {
        return this.mPaintVolume;
    }

    public void setPinchingProbability(double pinchingProbability) {
        this.mPinchingProbability = pinchingProbability;
    }

    public double getPinchingProbability() {
        return this.mPinchingProbability;
    }

    @Override
    public String print(int numPaper) {
        String resultOfPrint = this.mState.onPrint(numPaper);
        System.out.println(resultOfPrint);
        return resultOfPrint;
    }
    @Override
    public void loadPaper(int numPaper) {
        System.out.println(this.mState.onLoadPaper(numPaper));
    }
    @Override
    public void extractClampingPaper() {
        System.out.println(this.mState.onExtractClampingPaper());
    }
    @Override
    public void refilleCartridge() {
        System.out.println(this.mState.onRefilleCartridge());
    }

}
```

```
import java.util.Random;

class Main
{
    public static void main(String[] args)
    {
        Printer printer = new Printer("Canon 257B", 30, 100, 0.5);

        int numPaper;
        String result;
        Random r = new Random();

        for(int i = 0; i < 10; ++i)
```

```
        {
            numPaper = 0 + (10 - 0) * r.nextInt();
            printer.print(numPaper);

            numPaper = 0 + (10 - 0) * r.nextInt();
            result = printer.print(numPaper);

            if(result.lastIndexOf("Этап захвата бумаги.") != -1) {
                printer.extractClampingPaper();
            }

            if(result.lastIndexOf("Недостаточное количество бумаги") != -1) {
                numPaper = 0 + (10 - 0) * r.nextInt();
                printer.loadPaper(numPaper);
            }

            if(result.lastIndexOf("Недостаточное количество чернил") != -1) {
                printer.refilleCartridge();
            }
        }
    }
}
```

Пример работы программы

```
Состояние ожидания --> Этап печати.

Бумага напечатана.
Этап печати --> Состояние ожижания.

Состояние ожидания --> Этап печати.

Бумага не может быть распечатана. Подтверждено.
Этап печати --> Этап зажима бумаги.

Бумага не может быть распечатана.
Этап захвата бумаги.

Бумага не может быть распечатана.
Этап захвата бумаги.

Бумага извлечена.
Этап захвата бумаги --> Этап ожидания.

Состояние ожидания --> Этап печати.

Бумага напечатана.
Этап печати --> Состояние ожижания.

Состояние ожидания --> Этап печати.

Остальная бумага не может быть распечатана. Недостаточно краски в картридже.
Этап печати --> Состояние сбоя.

Остальная бумага не может быть распечатана. Недостаточно краски в картридже.
Состояние сбоя.
```

Вывод: приобрёл навыки применения паттернов проектирования при решении практических задач с использованием языка Java.