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

Лабораторная работа №6
По дисциплине: «ССП»
Вариант - 12

Выполнил:
Студент 3 курса
Группы ПО-8
Иванюк М.С.
Проверил:
Крощенко А.А

Брест, 2024

Лабораторная работа №6

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

Последняя цифра в зачетной книжке – 6.

Задание 1: Музыкальный магазин. Должно обеспечиваться одновременное обслуживание нескольких покупателей. Магазин должен предоставлять широкий выбор товаров различных музыкальных направлений**.**

Для музыкального магазина можно использовать паттерн Singleton, поскольку музыкальный магазин должен представлять собой единую систему, где управление товарами и покупками происходит в рамках одного магазина. Использование Singleton позволяет гарантировать, что в приложении будет только один экземпляр магазина, и обеспечивает удобный способ доступа к этому экземпляру из любой точки программы.

Код программы:

```
package lab6;

import java.util.HashMap;
import java.util.Map;
import java.util.Objects;

public class Task1{
    public static void main(String[] args){
        MusicStore store = MusicStore.getInstance();
        addMusicItems(store);
        User user1 = new User("Dima");
        User user2 = new User("Kirill");
        User user3 = new User("Stas");

        user1.buyMusicItem(store,"Rock",new MusicItem("Numb", "Linkin
Park"));
        System.out.println();
        user2.buyMusicItem(store,"Rock",new MusicItem("Numb", "Linkin
Park"));
        System.out.println();
        user2.buyMusicItem(store,"Rock",new MusicItem("In the End", "Linkin
Park"));
        System.out.println();
        user3.buyMusicItem(store,"Jazz",new MusicItem("Take Five", "Dave
Brubeck"));
        System.out.println();
        store.displayMusicItems();
    }
    public static void addMusicItems(MusicStore store){
        store.addMusicItem("Rock", new MusicItem("Numb", "Linkin Park"), 1);
        store.addMusicItem("Rock", new MusicItem("In the End", "Linkin
Park"), 15);
        store.addMusicItem("Rock", new MusicItem("Breaking the Habit",
"Linkin Park"), 10);
        store.addMusicItem("Pop", new MusicItem("Shape of You", "Ed
Sheeran"), 30);
        store.addMusicItem("Pop", new MusicItem("Someone Like You", "Adele"),
25);
```

```
        store.addMusicItem("Jazz", new MusicItem("Take Five", "Dave
Brubeck"), 12);
        store.addMusicItem("Jazz", new MusicItem("So What", "Miles Davis"),
8);
    }
}

class MusicStore{
    private static MusicStore uniqueInstance;
    private Map<String, Map<MusicItem, Integer>> musicItemsMap;
    MusicStore(){
        this.musicItemsMap = new HashMap<>();
    }
    public static synchronized MusicStore getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new MusicStore();
        }
        return uniqueInstance;
    }

    public void addMusicItem(String category,MusicItem item, int quantity){
        this.musicItemsMap.computeIfAbsent(category,k->new
HashMap<>()).put(item,quantity);
        //System.out.println("Товар добавлен в каталог.");
    }

    public boolean buyMusicItem(String category, MusicItem item) {
        if (!musicItemsMap.containsKey(category) ||
!musicItemsMap.get(category).containsKey(item) ||
musicItemsMap.get(category).get(item) <= 0) {
            System.out.println("Товар " + item.getTitle() + " - " +
item.getArtist() + " отсутствует в наличии.");
            return false;
        }
        int remainingQuantity = musicItemsMap.get(category).get(item) - 1;
        musicItemsMap.get(category).put(item, remainingQuantity);
        System.out.println("Покупка товара " + item.getTitle() + " - " +
item.getArtist() + ", Остаток: " + musicItemsMap.get(category).get(item));
        return true;
    }
    public void displayMusicItems() {
        System.out.println("В наличии:");
        for (Map.Entry<String, Map<MusicItem, Integer>> categoryEntry :
musicItemsMap.entrySet()) {
            String category = categoryEntry.getKey();
            System.out.println("\tКатегория: " + category);
            for (Map.Entry<MusicItem, Integer> itemEntry :
categoryEntry.getValue().entrySet()) {
                MusicItem item = itemEntry.getKey();
                int quantity = itemEntry.getValue();
                System.out.println("\t\t\tНазвание: " + item.getTitle() + ",
Исполнитель: " + item.getArtist() + ", Количесвто: " + quantity);
            }
        }
    }
}

class MusicItem{
    private String title;
    private String artist;

    public MusicItem(String title, String artist) {
        this.title = title;
        this.artist = artist;
```

```
    }

    public String getTitle() {
        return title;
    }

    public String getArtist() {
        return artist;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void setArtist(String artist) {
        this.artist = artist;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MusicItem musicItem = (MusicItem) o;
        return Objects.equals(title, musicItem.title) &&
Objects.equals(artist, musicItem.artist);
    }

    @Override
    public int hashCode() {
        return Objects.hash(title, artist);
    }

    @Override
    public String toString() {
        return "MusicItem{" +
                "title='" + title + '\'' +
                ", artist='" + artist + '\'' +
                '}';
    }
}

class User{
    private String name;

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void buyMusicItem(MusicStore store, String category, MusicItem
item) {
        System.out.println(name + " пытается купить " + item.getTitle() + " -
" + item.getArtist());
        boolean success = store.buyMusicItem(category, item);
        if (!success) {
            System.out.println("Покупка не удалась для " + item.getTitle() +
" - " + item.getArtist());
```

```
        }
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                '}';
    }
}
```

Результат работы программы:

```
"D:\Porgramming instruments\JDK2023\Java\jdk-21\bin\java.exe" "-javaagent:D:\Porgrammi
Dima пытается купить Numb - Linkin Park
Покупка товара Numb - Linkin Park, Остаток: 0

Kirill пытается купить Numb - Linkin Park
Товар Numb - Linkin Park отсутствует в наличии.
Покупка не удалась для Numb - Linkin Park

Kirill пытается купить In the End - Linkin Park
Покупка товара In the End - Linkin Park, Остаток: 14

Stas пытается купить Take Five - Dave Brubeck
Покупка товара Take Five - Dave Brubeck, Остаток: 11

В наличии:
    Категория: Pop
            Название: Someone Like You, Исполнитель: Adele, Количесвто: 25
            Название: Shape of You, Исполнитель: Ed Sheeran, Количесвто: 30
    Категория: Rock
            Название: Breaking the Habit, Исполнитель: Linkin Park, Количесвто: 10
            Название: Numb, Исполнитель: Linkin Park, Количесвто: 0
            Название: In the End, Исполнитель: Linkin Park, Количесвто: 14
    Категория: Jazz
            Название: So What, Исполнитель: Miles Davis, Количесвто: 8
            Название: Take Five, Исполнитель: Dave Brubeck, Количесвто: 11
```

Задание 2: Учетная запись покупателя книжного интернет-магазина. Предусмотреть различные уровни учетки в зависимости от активности покупателя. Дополнительные уровни добавляют функциональные возможности и открывают доступ к уникальным предложениям.

Для данного задания можно использовать паттерн “Декоратор”, т.к данный паттерн позволяет добавлять новую функциональность объектом, не изменяя их основной структуры. В рамках задания это означает, что мы можем добавлять новые уровни учетных записей не затрагивая существующий код.

Код программы:

```
package lab6;

public class Task2 {
    public static void main(String[] args){
        String bookName = "Java. Developer's guide";
        int grade = 10;

        Client client = new Client();

        client.buyBook(bookName);
        client.addToFavourite(bookName);
        client.appreciateBook(bookName,grade);

        System.out.println();
        client.upgradeToStandardAccount();
        client.buyBook(bookName);
        client.addToFavourite(bookName);
        client.appreciateBook(bookName,grade);

        System.out.println();
        client.upgradeToPremiumAccount();
        client.buyBook(bookName);
        client.addToFavourite(bookName);
        client.appreciateBook(bookName,grade);

    }
}


//паттерн Декоратор
interface IAccount{
    void buyBook(String bookName);
    void addToFavourite(String bookName);
    void appreciateBook(String bookName,int grade);
}

class BasicAccount implements IAccount{
    private String accountType;

    BasicAccount(){
        this.accountType = "Базовый аккаунт";
    }
    @Override
    public void buyBook(String bookName) {
        System.out.println(String.format("Вы купили книгу %s",bookName));
    }

    @Override
    public void addToFavourite(String bookName) {
        System.out.println("Вы не можете добавить в избранное через базовый
аккаунт.");
    }

    @Override
    public void appreciateBook(String bookName, int grade) {
```

```
        System.out.println("Вы не можете оценить книгу через базовый
аккаунт.");
    }
}

abstract class AccountDecorator implements IAccount{
    protected IAccount decoratedAccount;
    protected String accountType;

    public AccountDecorator(IAccount decoratedAccount) {
        this.decoratedAccount = decoratedAccount;
    }

    @Override
    public void buyBook(String bookName) {
        this.decoratedAccount.buyBook(bookName);
    }

    @Override
    public void addToFavourite(String bookName) {
        this.decoratedAccount.addToFavourite(bookName);
    }

    @Override
    public void appreciateBook(String bookName, int grade) {
        this.decoratedAccount.appreciateBook(bookName,grade);
    }
}

class StandardAccount extends AccountDecorator{

    public StandardAccount(IAccount decoratedAccount) {
        super(decoratedAccount);
        this.accountType = "Стандартный аккаунт";
    }

    @Override
    public void buyBook(String bookName) {
        System.out.println(String.format("Вы купили книгу %s",bookName));
    }

    @Override
    public void addToFavourite(String bookName) {
        System.out.println(String.format("Вы добавили  книгу %s в
избранное.",bookName));
    }

    @Override
    public void appreciateBook(String bookName, int grade) {
        System.out.println("Вы не можете оценить книгу через стандартный
аккаунт.");
    }
}

class PremiumAccount extends AccountDecorator{

    public PremiumAccount(IAccount decoratedAccount) {
        super(decoratedAccount);
        this.accountType = "Премиум аккаунт";
    }

    @Override
    public void buyBook(String bookName) {
        System.out.println(String.format("Вы купили книгу %s",bookName));
```

```
    }

    @Override
    public void addToFavourite(String bookName) {
        System.out.println(String.format("Вы добавили  книгу %s в
избранное.",bookName));
    }

    @Override
    public void appreciateBook(String bookName, int grade) {
        System.out.println(String.format("Вы оценили книгу %s на оценку
%d.",bookName,grade));
    }
}

class Client{
    private IAccount accountType;

    Client(){
        this.accountType = new BasicAccount();
    }

    public void upgradeToStandardAccount(){
        System.out.println("Повышение учетной записи до стандартного
уровня...");
        this.accountType = new StandardAccount(this.accountType);
        System.out.println("Ваша учетная запись успешно обновлена до
стандартного уровня!");
    }

    public void upgradeToPremiumAccount(){
        System.out.println("Повышение учетной записи до премиум уровня...");
        this.accountType = new PremiumAccount(this.accountType);
        System.out.println("Поздравляем! Ваша учетная запись теперь
премиум!");
    }

    public void addToFavourite(String bookName){
         this.accountType.addToFavourite(bookName);
    }

    public void buyBook(String bookName){
         this.accountType.buyBook(bookName);
    }

    public void appreciateBook(String bookName, int grade){
         this.accountType.appreciateBook(bookName,grade);
    }
}
```

Результат работы программы:

```
"D:\Porgramming instruments\JDK2023\Java\jdk-21\bin\java.exe" "
Вы купили книгу Java. Developer's guide
Вы не можете добавить в избранное через базовый аккаунт.
Вы не можете оценить книгу через базовый аккаунт.

Повышение учетной записи до стандартного уровня...
Ваша учетная запись успешно обновлена до стандартного уровня!
Вы купили книгу Java. Developer's guide
Вы добавили  книгу Java. Developer's guide в избранное.
Вы не можете оценить книгу через стандартный аккаунт.

Повышение учетной записи до премиум уровня...
Поздравляем! Ваша учетная запись теперь премиум!
Вы купили книгу Java. Developer's guide
Вы добавили  книгу Java. Developer's guide в избранное.
Вы оценили книгу Java. Developer's guide на оценку 10.
```

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.

Для проекта "Принтер" можно использовать паттерн проектирования "Состояние" (State), который позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. Этот паттерн хорошо подходит для моделирования различных режимов работы принтера: ожидание, печать документа, зажатие бумаги и отказ.

Код программы:

```
package lab6;
import java.util.Random;

public class Task3 {
    public static void main(String[] args) {
        Document document = new Document("+3752914880101.");
        Printer printer = new Printer("HP",100,80,1);
        printer.printDocument(document);

        System.out.println("\n");
        printer.setInkPercentage(0);
        printer.printDocument(document);
        printer.cartridgeRefilling(20);
        printer.printDocument(document);


        System.out.println("\n");
        printer.extractJammedPaper();
        printer.printDocument(document);
```

```
    }
}


// паттерн "Состояние"
interface PrinterState{
    void printDocument(Printer printer,Document document);
    void loadPaper(Printer printer,int count);
    void extractJammedPaper(Printer printer);
    void cartridgeRefilling(Printer printer,double percentage);
}

class PrinterWaitingState implements PrinterState{

    @Override
    public void printDocument(Printer printer, Document document) {
        System.out.println("Принтер готов к печати.");
        printer.changeState(new PrinterPrintingState());
        printer.printDocument(document);
    }

    @Override
    public void loadPaper(Printer printer, int count) {
        printer.setPaperCount(count);
        System.out.println("В принтер добавлено " + count + " листов
бумаги.");
    }

    @Override
    public void extractJammedPaper(Printer printer) {
        System.out.println("На данный момент нет зажатой бумаги.");
    }

    @Override
    public void cartridgeRefilling(Printer printer, double percentage) {
        printer.setInkPercentage(percentage);
        System.out.println("Катридж заправлен на " + percentage + " %.");
    }
}

class PrinterPrintingState implements PrinterState{

    @Override
    public void printDocument(Printer printer, Document document) {
        Random rnd = new Random();


        if(printer.getPaperCount() > 0 && printer.getInkPercentage() > 0){
            System.out.println("Печать документа, с содержимым: " +
document.getContent());

            double possibilityOfJamming = rnd.nextDouble()*100;
            if(possibilityOfJamming < printer.getPossibilityOfJamming()){
                System.out.println("Зажатие бумаги...");
                printer.changeState(new PrinterPaperJamState());
            }
            else{
                System.out.println("Документ успешно распечатан. Принтер в
состоянии ожидания...");
                printer.changeState(new PrinterWaitingState());
```

```
            }

        }
        else{
            System.out.println("В принтере закончилась краска или
бумага...");
            printer.changeState(new PrinterFailureState());
        }
    }

    @Override
    public void loadPaper(Printer printer, int count) {
        System.out.println("Нельзя загрузить бумагу когда принтер
печает...");
    }

    @Override
    public void extractJammedPaper(Printer printer) {
        System.out.println("Нельзя извелечь бумагу когда принтер печает...");
    }

    @Override
    public void cartridgeRefilling(Printer printer, double percentage) {
        System.out.println("Нельзя заправить катридж когда принтер
печает...");
    }
}

class PrinterPaperJamState implements PrinterState{

    @Override
    public void printDocument(Printer printer, Document document) {
        System.out.println("В принтере есть зажатая бумага, извеликинете
зажатую бумагу для продолжения печати...");
    }

    @Override
    public void loadPaper(Printer printer, int count) {
        printer.setPaperCount(count);
        System.out.println("В принтер добавлено " + count + " листов
бумаги.");
        printer.changeState(new PrinterWaitingState());
    }

    @Override
    public void extractJammedPaper(Printer printer) {
        System.out.println("Извелечение зажатой бумаги...");
        printer.changeState(new PrinterWaitingState());
    }

    @Override
    public void cartridgeRefilling(Printer printer, double percentage) {
        System.out.println("Извлеките зажатую бумагу прежде чем заправить
катридж краской...");
    }
}

class PrinterFailureState implements PrinterState{

    @Override
    public void printDocument(Printer printer, Document document) {
        System.out.println("В принетере закончилась краска или бумага...");
    }
```

```
    @Override
    public void loadPaper(Printer printer, int count) {
        printer.setPaperCount(count);
        System.out.println("В принтер добавлено " + count + " листов
бумаги.");
        printer.changeState(new PrinterWaitingState());
    }

    @Override
    public void extractJammedPaper(Printer printer) {
        System.out.println("На данный момент в принтеере отсуствует зажатая
бумага...");
    }

    @Override
    public void cartridgeRefilling(Printer printer, double percentage) {
        printer.setInkPercentage(percentage);
        System.out.println("Катридж заправлен на " + percentage + " %.");
        printer.changeState(new PrinterWaitingState());
    }
}

class Printer{

    private PrinterState currentState;
    private String model;
    private int paperCount;
    private double inkPercentage; //% краски в картридже
    private double possibilityOfJamming; //вероятность зажатия

    public Printer(String model, int paperCount, double inkPercentage, double
possibilityOfJamming) {
        this.model = model;
        this.paperCount = paperCount;
        this.inkPercentage = inkPercentage;
        this.possibilityOfJamming = possibilityOfJamming;
        this.currentState = new PrinterWaitingState();
    }

    void printDocument(Document document){
        this.currentState.printDocument(this,document);
    }
    void loadPaper(int paperCount){
        this.currentState.loadPaper(this,paperCount);
    }

    void extractJammedPaper(){
        this.currentState.extractJammedPaper(this);
    }
    void cartridgeRefilling(double inkPercentage){
        this.currentState.cartridgeRefilling(this,inkPercentage);
    }

    void changeState(PrinterState state){
        this.currentState = state;
    }

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

    public void setPaperCount(int paperCount) {
        this.paperCount = paperCount;
    }
```

```
    public void setInkPercentage(double inkPercentage) {
        this.inkPercentage = inkPercentage;
    }

    public void setPossibilityOfJamming(double possibilityOfJamming) {
        this.possibilityOfJamming = possibilityOfJamming;
    }

    public String getModel() {
        return model;
    }

    public int getPaperCount() {
        return paperCount;
    }

    public double getInkPercentage() {
        return inkPercentage;
    }

    public double getPossibilityOfJamming() {
        return possibilityOfJamming;
    }
}
class Document{
    private String content;

    Document(String content){
        this.content = content;
    }

    String getContent(){
        return this.content;
    }

    void setContent(String content){
        this.content = content;
    }

    @Override
    public String toString() {
        return String.format("Содержимое документа: \n" + this.content);
    }
}
```

Результат работы программы:

```
"D:\Porgramming instruments\JDK2023\Java\jdk-21\bin\java.exe
Принтер готов к печати.
Печать документа, с содержимым: +3752914880101.
Документ успешно распечатан. Принтер в состоянии ожидания...


Принтер готов к печати.
В принтере закончилась краска или бумага...
Катридж заправлен на 20.0 %.
Принтер готов к печати.
Печать документа, с содержимым: +3752914880101.
Документ успешно распечатан. Принтер в состоянии ожидания...


На данный момент нет зажатой бумаги.
Принтер готов к печати.
Печать документа, с содержимым: +3752914880101.
Документ успешно распечатан. Принтер в состоянии ожидания...
```

Вывод: в ходе выполнения лабораторной работы я приобрел навыки применения паттернов проектирования при решении практических задач с использованием языка Java.