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

Отчёт
по лабораторной работе №6

Выполнил:
студент группы ПО-9
Зеленков К. И.

Проверил:
Крощенко А. А.

Брест 2024

Вариант 6

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

Задание 1

Музыкальный магазин. Должно обеспечиваться одновременное обслуживание нескольких покупателей. Магазин должен предоставлять широкий выбор товаров. Выбранный Паттерн: Наблюдатель. Он позволяет реализовать механизм уведомления об изменениях в состоянии объекта одному или нескольким зависимым объектам. У нас есть несколько покупателей, которые должны быть уведомлены об изменениях в ассортименте товаров (новые альбомы, специальные предложения и т. д.)

Код программы:

Main1.java:

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

interface Observer {
    void update(News news);
}

class News {
    private String text;

    public News(String text) {
        this.text = text;
    }

    public String getText() {
        return text;
    }
}

class MusicStore {
    private List<Observer> observers = new ArrayList<>();
    private Map<String, Double> availableAlbums = new HashMap<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(News news) {
        for (Observer observer : observers) {
            observer.update(news);
        }
    }

    public void addNews(String newsText) {
        News news = new News(newsText);
        notifyObservers(news);
    }

    public void addAlbum(String albumName, double price) {
```

```
        availableAlbums.put(albumName, price);
    }

    public void purchaseAlbum(Customer customer, String albumName) {
        if (availableAlbums.containsKey(albumName)) {
            double price = availableAlbums.get(albumName);
            if (customer.hasEnoughBalance(price)) {
                customer.deductBalance(price);
                availableAlbums.remove(albumName);
                System.out.println(customer.getName() + " приобрел " +
albumName + " за $" + price);
            } else {
                System.out.println(customer.getName() + " недостаточно
средств для покупки " + albumName);
            }
        } else {
            System.out.println("Альбом " + albumName + " не доступен в
магазине.");
        }
    }
}

class Customer implements Observer {
    private String name;
    private double balance;

    public Customer(String name, double balance) {
        this.name = name;
        this.balance = balance;
    }

    @Override
    public void update(News news) {
        System.out.println(name + " узнал о новостях: " + news.getText());
    }

    public String getName() {
        return name;
    }

    public double getBalance() {
        return balance;
    }

    public boolean hasEnoughBalance(double amount) {
        return balance >= amount;
    }

    public void deductBalance(double amount) {
        balance -= amount;
    }
}

class Main1 {
    public static void main(String[] args) {
        MusicStore store = new MusicStore();

        Customer customer1 = new Customer("Костя", 120);
        Customer customer2 = new Customer("Андрей", 30);

        store.addObserver(customer1);
        store.addObserver(customer2);

        store.addAlbum("Альбом 1", 25.0);
        store.addAlbum("Альбом 2", 35.0);
        store.addAlbum("Альбом 3", 20.0);
```

```
        store.addNews("В магазине появились новые альбомы!");

        store.purchaseAlbum(customer1, "Альбом 1");
        store.purchaseAlbum(customer2, "Альбом 2");
        store.purchaseAlbum(customer2, "Альбом 3");
        store.purchaseAlbum(customer1, "Альбом 2");
    }
}
```

Результат работы программы:

```
Костя узнал о новостях: В магазине появились новые альбомы!
Андрей узнал о новостях: В магазине появились новые альбомы!
Костя приобрел Альбом 1 за $25.0
Андрей недостаточно средств для покупки Альбом 2
Андрей приобрел Альбом 3 за $20.0
Костя приобрел Альбом 2 за $35.0
```

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.

Был выбран паттерн "Декоратор", так как он подходит лучше для ситуаций, когда требуется добавить дополнительную функциональность к объекту, сохраняя его базовый интерфейс и структуру. Базовый уровень учетной записи может только получать очки и улучшаться до премиума, премиум может участвовать в розыгрыше с призом в $10, а также получить элитный уровень, а элитный, имеет функционал розыгрыша с призом в $15, а также кэшбэк с каждой купленной книги в виде 10% от стоимости книги.

Код программы

Main2.java:

```
import java.util.Random;
import java.util.Random;

class PremiumLevelDecorator extends BasicLevelDecorator {
    @Override
    public void purchaseBook(Book book, CustomerAccount customer) {
        super.purchaseBook(book, customer);
    }

    @Override
    public void checkUpgrade(CustomerAccount customer) {
        if (customer.getPoints() >= 10) {
            customer.setActivityLevel(new EliteLevelDecorator());
            System.out.println("Поздравляю! Вы были повышены до элитного
уровня.");
        }
    }
    public void participateInContest(CustomerAccount customer) {
        Random random = new Random();
```

```
        int result = random.nextInt(100);
        if (result < 30) {
            System.out.println("Вы выиграли конкурс!");
            customer.addBalance(10);
        } else {
            System.out.println("На этот раз никаких призов. В следующий раз
точно повезет!");
        }
    }
}


class EliteLevelDecorator extends PremiumLevelDecorator {
    @Override
    public void purchaseBook(Book book, CustomerAccount customer) {
        System.out.println("Купленная книга: " + book.getTitle());
        customer.incrementPoints(book.getPoints());
        double discount = book.getPrice() * 0.1; // 10% скидка для элиты
        customer.addBalance(discount);
        System.out.println("Вы получили кэшбэк в размере $" + discount + "
для вашего элитного уровня.");
    }

    @Override
    public void participateInContest(CustomerAccount customer) {
        Random random = new Random();
        int result = random.nextInt(100);
        if (result < 30) {
            System.out.println("Вы выиграли конкурс!");
            customer.addBalance(15);
        } else {
            System.out.println("На этот раз никаких призов. В следующий раз
точно повезет!");
        }
    }
}

interface ActivityLevel {
    void purchaseBook(Book book, CustomerAccount customer);
    void checkUpgrade(CustomerAccount customer);
    void participateInContest(CustomerAccount customer);
}

class BasicLevelDecorator implements ActivityLevel {
    @Override
    public void purchaseBook(Book book, CustomerAccount customer) {
        System.out.println("Купленная книга: " + book.getTitle());
        customer.incrementPoints(book.getPoints());
        checkUpgrade(customer);
    }

    @Override
    public void checkUpgrade(CustomerAccount customer) {
        if (customer.getPoints() >= 5) {
            customer.setActivityLevel(new PremiumLevelDecorator());
            System.out.println("Поздравляю! Вы были повышены до премиум-
уровня.");
        }
    }

    @Override
    public void participateInContest(CustomerAccount customer){
        System.out.println("Участие в конкурсе недоступно для текущего
уровня активности.");
    }
}
```

```
class Book {
    private String title;
    private int points;
    private double price;

    public Book(String title, int points, double price) {
        this.title = title;
        this.points = points;
        this.price = price;
    }

    public String getTitle() {
        return title;
    }

    public int getPoints() {
        return points;
    }

    public double getPrice() {
        return price;
    }
}

class CustomerAccount {
    private ActivityLevel activityLevel;
    private int points;
    private double balance;

    public CustomerAccount() {
        this.activityLevel = new BasicLevelDecorator();
        this.points = 0;
        this.balance = 0;
    }

    public void setActivityLevel(ActivityLevel activityLevel) {
        this.activityLevel = activityLevel;
    }

    public void incrementPoints(int points) {
        this.points += points;
    }

    public int getPoints() {
        return points;
    }

    public double getBalance() {
        return balance;
    }

    public void purchaseBook(Book book) {
        if (book.getPrice() <= balance) {
            System.out.println("Купленная книга: " + book.getTitle() + " за
$" +
                    book.getPrice());
            balance -= book.getPrice();
            System.out.println("Оставшийся баланс: $" + balance);
            activityLevel.purchaseBook(book, this);
        } else {
            System.out.println("Недостаточно средств для покупки " +
book.getTitle());
        }
    }
```

```
    public void participateInContest() {
        activityLevel.participateInContest(this);
    }

    public void addBalance(double amount) {
        balance += amount;
        System.out.println("Добавлено $" + amount + " на ваш баланс. Текущий
баланс: $" + balance);
    }
}

public class Main2 {
    public static void main(String[] args) {
        CustomerAccount customer1 = new CustomerAccount();
        Book book1 = new Book("Книга 1", 2, 10.0);
        Book book2 = new Book("Книга 2", 3, 15.0);
        Book book3 = new Book("Книга 3", 4, 20.0);
        Book book4 = new Book("Книга 4", 2, 30.0);
        customer1.addBalance(100.0);

        customer1.purchaseBook(book1);
        customer1.purchaseBook(book2);
        customer1.purchaseBook(book3);

        System.out.println("Очки клиента: " + customer1.getPoints());
        System.out.println("Баланс клиента: $" + customer1.getBalance());

        customer1.addBalance(50.0);

        customer1.participateInContest();

        customer1.purchaseBook(book2);

        customer1.purchaseBook(book4);


        customer1.participateInContest();
    }
}
```

Результат работы программы:

```
Добавлено $100.0 на ваш баланс. Текущий баланс: $100.0
Купленная книга: Книга 1 за $15.0
Оставшийся баланс: $85.0
Купленная книга: Книга 1
Купленная книга: Книга 2 за $10.0
Оставшийся баланс: $75.0
Купленная книга: Книга 2
Поздравляю! Вы были повышены до премиум-уровня.
Купленная книга: Книга 3 за $25.0
Оставшийся баланс: $50.0
Купленная книга: Книга 3
Очки клиента: 9
Баланс клиента: $50.0
Добавлено $50.0 на ваш баланс. Текущий баланс: $100.0
Вы выиграли конкурс!
Добавлено $10.0 на ваш баланс. Текущий баланс: $110.0
Купленная книга: Книга 2 за $10.0
Оставшийся баланс: $100.0
Купленная книга: Книга 2
Поздравляю! Вы были повышены до элитного уровня.
Купленная книга: Книга 4 за $20.0
Оставшийся баланс: $80.0
Купленная книга: Книга 4
Добавлено $2.0 на ваш баланс. Текущий баланс: $82.0
Вы получили кэшбэк в размере $2.0 для вашего элитного уровня.
На этот раз никаких призов. В следующий раз точно повезет!
```

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. Был выбран паттерн State (состояний) для реализации проекта "Принтер". Паттерн состояний позволяет объекту изменять свое поведение в зависимости от внутреннего состояния.

Код программы

Main3.java:

```
import java.util.Scanner;
import java.util.Random;

interface PrinterState {
    void printDocument(Printer printer);
    void removeJam(Printer printer);
    void loadPaper(Printer printer, int count);
    void refillInk(Printer printer);
    String getStateDescription();
}

class PrintingState implements PrinterState {
    @Override
    public void printDocument(Printer printer) {
        int jamProbability = new Random().nextInt(100) + 1;
        if (jamProbability <= printer.getJamProbability()) {
            System.out.println("Бумага замята!");
            printer.setCurrentState(new PaperJamState());

System.out.println(printer.getCurrentState().getStateDescription());
        } else if (printer.getPaperCount() <= 0 || printer.getInkLevel() <=
0) {
            System.out.println("Закончилась бумага или чернила. Печать
остановлена.");
            printer.setCurrentState(new OutOfPaperOrInkState());

System.out.println(printer.getCurrentState().getStateDescription());
        }
        else {
            System.out.println("Уже печатается...");
            printer.setPaperCount(printer.getPaperCount() - 1);
            printer.setInkLevel(printer.getInkLevel() - 10);
            printer.setCurrentState(new IdleState());

System.out.println(printer.getCurrentState().getStateDescription());
        }
    }

    @Override
    public void removeJam(Printer printer) {
        System.out.println("Не удается устранить замятие бумаги во время
печати.");
    }

    @Override
    public void loadPaper(Printer printer, int count) {
        System.out.println("Не удается загрузить бумагу во время печати.");
    }
```

```
    @Override
    public void refillInk(Printer printer) {
        System.out.println("Невозможно долить чернила во время печати.");
    }

    @Override
    public String getStateDescription() {
        return "Состояние печати";
    }
}

class IdleState implements PrinterState {
    @Override
    public void printDocument(Printer printer) {
        System.out.println("Переключение в режим печати...");
        printer.setCurrentState(new PrintingState());
        System.out.println(printer.getCurrentState().getStateDescription());
        printer.printDocument();
    }

    @Override
    public void removeJam(Printer printer) {
        System.out.println("Нет замятостей для устранения.");
    }

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

    @Override
    public void refillInk(Printer printer) {
        printer.setInkLevel(100);
        System.out.println("Чернила заправлены на 100%.");
    }

    @Override
    public String getStateDescription() {
        return "Состояние ожидания";
    }
}

class Printer {
    private String model;
    private int paperCount;
    private int inkLevel;
    private int jamProbability;
    private PrinterState currentState;

    public Printer(String model, int paperCount, int inkLevel, int
jamProbability) {
        this.model = model;
        this.paperCount = paperCount;
        this.inkLevel = inkLevel;
        this.jamProbability = jamProbability;
        this.currentState = new IdleState();
    }

    public void printDocument() {
        currentState.printDocument(this);
    }

    public void loadPaper(int count) {
        currentState.loadPaper(this, count);
```

```
    }

    public void refillInk() {
        currentState.refillInk(this);
    }

    public void removeJam() {
        currentState.removeJam(this);
    }

    public String getModel() {
        return model;
    }

    public int getPaperCount() {
        return paperCount;
    }

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

    public int getInkLevel() {
        return inkLevel;
    }

    public void setInkLevel(int inkLevel) {
        this.inkLevel = inkLevel;
    }

    public int getJamProbability() {
        return jamProbability;
    }

    public void setCurrentState(PrinterState currentState) {
        this.currentState = currentState;
    }

    public PrinterState getCurrentState() {
        return currentState;
    }

    public void printStatus() {
        System.out.println("Модель принтера: " + model);
        System.out.println("Кличество бумаги: " + paperCount);
        System.out.println("Уровень краски: " + inkLevel + "%");
        System.out.println("Текущее состояние: " +
currentState.getStateDescription());
    }
}


class PaperJamState implements PrinterState {
    @Override
    public void printDocument(Printer printer) {
        System.out.println("Не удается выполнить печать. Бумага замята.");
    }

    @Override
    public void removeJam(Printer printer) {
        System.out.println("Устранение замятия бумаги...");
        printer.setCurrentState(new IdleState());
        System.out.println(printer.getCurrentState().getStateDescription());
    }

    @Override
```

```
    public void loadPaper(Printer printer, int count) {
        printer.setPaperCount(printer.getPaperCount() + count);
        System.out.println("Загружено " + count + " листов бумаги.");
    }

    @Override
    public void refillInk(Printer printer) {
        printer.setInkLevel(100);
        System.out.println("Чернила заправлены на 100%.");
    }

    @Override
    public String getStateDescription() {
        return "Состояние замятия бумаги.";
    }
}

class OutOfPaperOrInkState implements PrinterState {
    @Override
    public void printDocument(Printer printer) {
        System.out.println("Не удается распечатать. Закончилась бумага или
чернила.");
    }

    @Override
    public void removeJam(Printer printer) {
        System.out.println("Нет замятостей для устранения.");
    }

    @Override
    public void loadPaper(Printer printer, int count) {
        printer.setPaperCount(printer.getPaperCount() + count);
        printer.setCurrentState(new IdleState());
        System.out.println("Бумага загружена. Принтер готов.");
        System.out.println(printer.getCurrentState().getStateDescription());
    }

    @Override
    public void refillInk(Printer printer) {
        printer.setInkLevel(100);
        printer.setCurrentState(new IdleState());
        System.out.println("Чернила заправлены. Принтер готов.");
        System.out.println(printer.getCurrentState().getStateDescription());
    }

    @Override
    public String getStateDescription() {
        return "Чернила или бумага закончились.";
    }
}

public class Main3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Printer printer = new Printer("HP LaserJet Pro", 50, 10, 5);

        boolean isRunning = true;
        while (isRunning) {
            System.out.println("========== Меню ==========");
            System.out.println("1. Печать документа");
            System.out.println("2. Загрузить бумагу");
            System.out.println("3. Устранить замятие бумаги");
            System.out.println("4. Заправить чернила");
            System.out.println("5. Проверить статус принтера");
            System.out.println("0. Выход");
            System.out.print("Ваш выбор: ");
```

```
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    printer.printDocument();
                    break;
                case 2:
                    System.out.print("Enter number of sheets to load: ");
                    int sheets = scanner.nextInt();
                    printer.loadPaper(sheets);
                    break;
                case 3:
                    printer.removeJam();
                    break;
                case 4:
                    printer.refillInk();
                    break;
                case 5:
                    printer.printStatus();
                    break;
                case 0:
                    isRunning = false;
                    System.out.println("Exit...");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid
option.");
            }
            System.out.println();
        }
        scanner.close();
    }
}
```

Результат работы программы:

Проверка состояния бумаги и печать.

```
========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 5
Модель принтера: HP LaserJet Pro
Кличество бумаги: 50
Уровень краски: 10%
Текущее состояние: Состояние ожидания

========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 1
Переключение в режим печати...
Состояние печати
Уже печатается...
Состояние ожидания
```

Проверяем состояние, краска закончилась. Заправим ее и проверим состояние еще раз.

```
========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 5
Модель принтера: HP LaserJet Pro
Кличество бумаги: 49
Уровень краски: 0%
Текущее состояние: Состояние ожидания

========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 4
Чернила заправлены на 100%.

========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 5
Модель принтера: HP LaserJet Pro
Кличество бумаги: 49
Уровень краски: 100%
Текущее состояние: Состояние ожидания
```

Спустя какое-то количество попыток бумага замялась, проверим статус принтера.

```
========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 1
Переключение в режим печати...
Состояние печати
Бумага замята!
Состояние замятия бумаги.

========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 5
Модель принтера: HP LaserJet Pro
Кличество бумаги: 36
Уровень краски: 70%
Текущее состояние: Состояние замятия бумаги.
```

Устраняем замятие бумаги и печатаем вновь. Всё хорошо.

```
========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 3
Устранение замятия бумаги...
Состояние ожидания

========== Меню ==========
1. Печать документа
2. Загрузить бумагу
3. Устранить замятие бумаги
4. Заправить чернила
5. Проверить статус принтера
0. Выход
Ваш выбор: 1
Переключение в режим печати...
Состояние печати
Уже печатается...
Состояние ожидания
```

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