МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №6

Выполнил:
студент 3 курса
группы ПО-9
Тусюк Т.В.

Проверил:
Крощенко А.А.

Брест 2024

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

Задание 1:

- 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.

1) Кофе-автомат с возможностью создания различных кофейных напитков (предусмотреть 5 классов наименований)

Выбранный паттерн проектирования: **FactoryMethod.**

В данном случае фабричный метод применяется для создания различных типов кофе. Классы ItalianCoffeeShop и AmericanCoffeeShop используют разные фабрики кофе (ItalianCoffeeFactory и AmericanCoffeeFactory соответственно), что позволяет каждому магазину предлагать свои уникальные варианты кофе, не зависимо от того, какие новые типы кофе будут добавлены в будущем.

Выполнение задания:

```
public abstract class Coffee {
    public void grindCoffee() {
        System.out.println("Перемалываем кофе");
    }

    public void makeCoffee() {
        System.out.println("Делаем кофе");
    }

    public void pourIntoCup() {
        System.out.println("Наливаем в чашку");
    }

    public abstract void prepareCoffee();
}

class ItalianStyleAmericano extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен итальянский американо");
    }
}

class ItalianStyleCappucino extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен итальянский капучино");
    }
}

class ItalianStyleCaffeLatte extends Coffee {
```

```
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен итальянский кафе латте");
    }
}

class ItalianStyleEspresso extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен итальянский эспрессо");
    }
}

class AmericanStyleAmericano extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен американский американо");
    }
}

class AmericanStyleCappucino extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен американский капучино");
    }
}

class AmericanStyleCaffeLatte extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен американский кафе латте");
    }
}

class AmericanStyleEspresso extends Coffee {
    @Override
    public void prepareCoffee() {
        System.out.println("Приготовлен американский эспрессо");
    }
}

enum CoffeeType {
    ESPRESSO,
    AMERICANO,
    CAFFE_LATTE,
    CAPPUCCINO
}

abstract class CoffeeFactory {
    public abstract Coffee createCoffee(CoffeeType type);
}

class ItalianCoffeeFactory extends CoffeeFactory {
    @Override
    public Coffee createCoffee(CoffeeType type) {
        Coffee coffee = null;
        switch (type) {
            case AMERICANO:
                coffee = new ItalianStyleAmericano();
                break;
            case ESPRESSO:
```

```
                coffee = new ItalianStyleEspresso();
                break;
            case CAPPUCCINO:
                coffee = new ItalianStyleCappucino();
                break;
            case CAFFE_LATTE:
                coffee = new ItalianStyleCaffeLatte();
                break;
        }
        return coffee;
    }
}

class AmericanCoffeeFactory extends CoffeeFactory {
    @Override
    public Coffee createCoffee(CoffeeType type) {
        Coffee coffee = null;
        switch (type) {
            case AMERICANO:
                coffee = new AmericanStyleAmericano();
                break;
            case ESPRESSO:
                coffee = new AmericanStyleEspresso();
                break;
            case CAPPUCCINO:
                coffee = new AmericanStyleCappucino();
                break;
            case CAFFE_LATTE:
                coffee = new AmericanStyleCaffeLatte();
                break;
        }
        return coffee;
    }
}

abstract class CoffeeShop {

    private final CoffeeFactory coffeeFactory;

    public CoffeeShop(CoffeeFactory coffeeFactory) {
        this.coffeeFactory = coffeeFactory;
    }

    public Coffee orderCoffee(CoffeeType type) {
        Coffee coffee = coffeeFactory.createCoffee(type);
        coffee.grindCoffee();
        coffee.makeCoffee();
        coffee.pourIntoCup();
        coffee.prepareCoffee();

        System.out.println("Вот ваш " + type.toString() + "! Спасибо, приходите
еще!");
        return coffee;
    }
}

class ItalianCoffeeShop extends CoffeeShop {

    public ItalianCoffeeShop(CoffeeFactory coffeeFactory) {
        super(coffeeFactory);
    }
```

```
}

class AmericanCoffeeShop extends CoffeeShop {
    public AmericanCoffeeShop(CoffeeFactory coffeeFactory) {
        super(coffeeFactory);
    }
}
 class Main {
    public static void main(String[] args) {
        CoffeeFactory italianCoffeeFactory = new ItalianCoffeeFactory();
        CoffeeShop italianCoffeeShop = new ItalianCoffeeShop(italianCoffeeFactory);
        italianCoffeeShop.orderCoffee(CoffeeType.CAFFE_LATTE);

        CoffeeFactory americanCoffeeFactory = new AmericanCoffeeFactory();
        CoffeeShop americanCoffeeShop = new
AmericanCoffeeShop(americanCoffeeFactory);
        americanCoffeeShop.orderCoffee(CoffeeType.CAFFE_LATTE);
    }
}
```

Результат:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-j
Перемалываем кофе
Делаем кофе
Наливаем в чашку
Приготовлен итальянский кафе латте
Вот ваш CAFFE_LATTE! Спасибо, приходите еще!
Перемалываем кофе
Делаем кофе
Наливаем в чашку
Приготовлен американский кафе латте
Вот ваш CAFFE_LATTE! Спасибо, приходите еще!
```

Задание 2:

1) Проект «Часы». В проекте должен быть реализован класс, который дает возможность пользоваться часами со стрелками так же, как и цифровыми часами. В классе «Часы со стрелками» хранятся повороты стрелок.

Паттерн "Фасад" позволяет создать унифицированный интерфейс для набора интерфейсов в подсистеме. В данном случае, мы можем создать класс-фасад, который будет предоставлять методы для работы с часами, как с цифровыми, скрывая внутреннюю реализацию с использованием часов со стрелками.

Выполнение задания:

```
class AnalogClock {
    private int hours;
    private int minutes;
    private int seconds;

    public void setClock(int hours, int minutes, int seconds) {
        this.hours = hours;
```

```
        this.minutes = minutes;
        this.seconds = seconds;
    }

    public void rotateHands() {
        // Логика поворота стрелок
    }

    // Методы для получения времени
    public int getHours() {
        return hours;
    }

    public int getMinutes() {
        return minutes;
    }

    public int getSeconds() {
        return seconds;
    }
}

// Фасад для работы с часами
class ClockFacade {
    private AnalogClock analogClock;

    public ClockFacade() {
        this.analogClock = new AnalogClock();
    }

    // Методы для установки времени
    public void setTime(int hours, int minutes, int seconds) {
        analogClock.setClock(hours, minutes, seconds);
    }

    // Методы для получения времени в цифровом формате
    public int getHours() {
        return analogClock.getHours();
    }

    public int getMinutes() {
        return analogClock.getMinutes();
    }

    public int getSeconds() {
        return analogClock.getSeconds();
    }
}

// Пример использования
public class Task_02 {
    public static void main(String[] args) {
        ClockFacade clock = new ClockFacade();

        // Устанавливаем время
        clock.setTime(12, 30, 0);

        // Получаем время в цифровом формате
        System.out.println("Time: " + clock.getHours() + ":" + clock.getMinutes() +
":" + clock.getSeconds());
```

```
    }
}
```

Результат:

```
"C:\Program Files\Java\jdk-21\bin\java.ex
Time: 12:30:0
```

Задание 3:

Проект «Клавиатура настраимаемого калькулятора». Цифровые и арифметические кнопки имеют фиксированную функцию, а остальные могут менять своё назначение.

Паттерн "Команда" позволяет инкапсулировать запрос на выполнение определенного действия или операции в отдельном объекте. Этот объект содержит всю необходимую информацию для выполнения запроса, включая ссылку на объект, который должен выполнить действие. Также он может быть настраиваемым, что позволяет динамически изменять поведение программы.

Выполнение задания(Паттерн “Команда”):

```
// Интерфейс команды
interface Command {
    void execute();
}

// Реализация команды для выполнения арифметической операции
class ArithmeticCommand implements Command {
    private Calculator calculator;
    private char operator;

    public ArithmeticCommand(Calculator calculator, char operator) {
        this.calculator = calculator;
        this.operator = operator;
    }

    @Override
    public void execute() {
        calculator.performOperation(operator);
    }
}

// Реализация команды для очистки экрана
class ClearCommand implements Command {
    private Calculator calculator;

    public ClearCommand(Calculator calculator) {
        this.calculator = calculator;
    }

    @Override
    public void execute() {
        calculator.clearScreen();
    }
}

// Класс клавиши
class Button {
    private Command command;
```

```
    public Button(Command command) {
        this.command = command;
    }

    public void click() {
        command.execute();
    }
}

// Класс калькулятора
class Calculator {
    // Методы для выполнения арифметических операций и очистки экрана
    public void performOperation(char operator) {
        System.out.println("Performing operation: " + operator);
        // Логика выполнения операции
    }

    public void clearScreen() {
        System.out.println("Clearing the screen");
        // Логика очистки экрана
    }
}

// Пример использования
public class Task_03 {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();

        // Создаем кнопку для выполнения арифметической операции
        Command addCommand = new ArithmeticCommand(calculator, '+');
        Button addButton = new Button(addCommand);

        // Создаем кнопку для очистки экрана
        Command clearCommand = new ClearCommand(calculator);
        Button clearButton = new Button(clearCommand);

        // Используем кнопки
        addButton.click(); // Выполнить сложение
        clearButton.click(); // Очистить экран
    }
}
```

Результат:

```
Performing operation: +
Clearing the screen
```

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