

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

ОТЧЁТ
по лабораторной работе №6

Выполнил:
студент группы ПО-9
Ступак Д.Р

Проверил:
Крощенко А. А.

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

Вариант 9.

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.

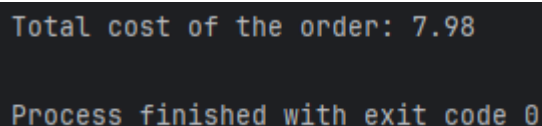
Задание 1.

Проект «Бургер-закусочная». Реализовать возможность формирования заказа из определенных позиций (тип бургера (веганский, куриный и т.д.)), напиток (холодный – пепси, кока-кола и т.д.; горячий – кофе, чай и т.д.), тип упаковки – с собой, на месте. Должна формироваться итоговая стоимость заказа.

Паттерн "Строитель" позволяет создавать сложные объекты шаг за шагом. Он позволяет отделить процесс конструирования объекта от его представления, что позволяет использовать один и тот же процесс конструирования для создания различных представлений объекта.

В данном случае, мы можем создать класс "Заказчик" (OrderBuilder), который будет отвечать за конструирование заказа. Для этого класса мы можем определить методы для выбора типа бургера, напитка, типа упаковки и расчета итоговой стоимости заказа. Класс "Заказчик" будет служить в роли строителя, который пошагово формирует заказ.

Работа программы:



```
Total cost of the order: 7.98

Process finished with exit code 0
```

Код программы:

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

class Order {
    private final List<Burger> burgers;
    private final List<Beverage> beverages;
    private final List<Packaging> packagings;

    public Order() {
        burgers = new ArrayList<>();
        beverages = new ArrayList<>();
        packagings = new ArrayList<>();
    }

    public void addBurger(Burger burger) {
        burgers.add(burger);
    }

    public void addBeverage(Beverage beverage) {
```

```

        beverages.add(beverage);
    }

    public void addPackaging(Packaging packaging) {
        packagings.add(packaging);
    }

    public double getTotalCost() {
        double totalCost = 0;

        for (Burger burger : burgers) {
            totalCost += burger.getPrice();
        }

        for (Beverage beverage : beverages) {
            totalCost += beverage.getPrice();
        }

        for (Packaging packaging : packagings) {
            totalCost += packaging.getPrice();
        }

        return totalCost;
    }
}

class Burger {
    public enum Type{
        Vegan(4.99),
        Chicken(9.99);
        private final Double price;
        Type(Double price){
            this.price = price;
        }
    }

    private final double price;

    public Burger(Type type) {
        this.price = type.price;
    }

    public double getPrice() {
        return price;
    }
}

class Beverage {

    public enum Type{
        pepsi(0.99),
        tea(1.5),
        coffee(4.9),
        fanta(2.99),
        sprite(3.5);
        private final Double price;
        Type(Double price){
            this.price = price;
        }
    }

    private final double price;

    public Beverage(Type type) {
        this.price = type.price;
    }

    public double getPrice() {

```

```

        return price;
    }
}
class Packaging {
    enum Type{
        here(0.0),
        yourself(1.0);
        private final Double price;
        Type(Double price){
            this.price = price;
        }
    }
    private final double price;

    public Packaging(Type type) {
        this.price = type.price;
    }

    public double getPrice() {
        return price;
    }
}
interface OrderBuilder {
    void chooseBurger(Burger.Type type);
    void chooseBeverage(Beverage.Type type);
    void choosePackaging(Packaging.Type type);
    Order getOrder();
}
class Cashier implements OrderBuilder {
    private Order order;

    public Cashier() {
        order = new Order();
    }

    @Override
    public void chooseBurger(Burger.Type type) {
        // Burger selection logic
        Burger burger = new Burger(type);
        order.addBurger(burger);
    }

    @Override
    public void chooseBeverage(Beverage.Type type) {
        // Beverage selection logic
        Beverage beverage = new Beverage(type);
        order.addBeverage(beverage);
    }

    @Override
    public void choosePackaging(Packaging.Type type) {
        // Packaging selection logic
        Packaging packaging = new Packaging(type);
        order.addPackaging(packaging);
    }

    @Override
    public Order getOrder() {
        return order;
    }
}
public class Lab6_1 {
    public static void main(String[] args) {
        Cashier cashier = new Cashier();
        cashier.chooseBurger(Burger.Type.Vegan);
    }
}

```

```

        cashier.chooseBeverage(Beverage.Type.fanta);
        cashier.choosePackaging(Packaging.Type.here);

        Order order = cashier.getOrder();
        double totalCost = order.getTotalCost();

        System.out.println("Total cost of the order: " + totalCost);
    }
}

```

Задание 2.

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 "Адаптер" позволяет объектам с несовместимыми интерфейсами работать вместе. Он оборачивает один интерфейс в другой, делая их совместимыми.

В данном случае, мы можем создать класс "Цифровые часы" (DigitalClock), который имеет интерфейс для работы с цифровыми часами. Затем мы можем создать класс "Часы со стрелками" (AnalogClock), который хранит информацию о поворотах стрелок. Для того, чтобы "Часы со стрелками" могли работать так же, как и цифровые часы, мы можем создать адаптер "Аналогово-цифровых часов" (AnalogToDigitalClockAdapter), который оборачивает объект "Часы со стрелками" и предоставляет интерфейс, совместимый с интерфейсом цифровых часов.

Работа программы:

```

Degrees of rotation of the arrows: 310 hours,45 seconds
Digital Clock (Adapter): 10:20:07

```

Код программы:

```

interface DigitalClock {
    void displayTime();
}
class DigitalClockImpl implements DigitalClock {
    @Override
    public void displayTime() {
        // Реализация вывода времени на цифровых часах
        System.out.println("Digital Clock: HH:MM:SS");
    }
}
class AnalogClock {
    private int hourDegree;

    private int secondDegree;

    public void setClock(int hour,int second) {
        // Установка времени на часах со стрелками
        this.hourDegree = hour;
        this.secondDegree = second;
    }

    public int getHourDegree() {
        return hourDegree;
    }

    public int getSecondDegree() {
        return secondDegree;
    }
}

```

```

    }
}
class AnalogToDigitalClockAdapter implements DigitalClock {
    private AnalogClock analogClock;
    public AnalogToDigitalClockAdapter(AnalogClock analogClock) {
        this.analogClock = analogClock;
    }
    @Override
    public void displayTime() {
        int hour = analogClock.getHourDegree()/30;
        double tempMinute= (double) analogClock.getHourDegree() /30;
        int minute = (int)((tempMinute-hour) *60);
        int second = analogClock.getSecondDegree()/6;

        System.out.println("Digital Clock (Adapter): " + String.format("%02d:%02d:%02d", hour,
minute, second));
    }
}
public class Lab6_2 {
    public static void main(String[] args) {

        AnalogClock analogClock = new AnalogClock();
        System.out.println("Degrees of rotation of the arrows: 310 hours,45 seconds");
        analogClock.setClock(310, 45);

        DigitalClock digitalClock = new AnalogToDigitalClockAdapter(analogClock);

        digitalClock.displayTime();
    }
}

```

Задание 3.

Шифрование текстового файла. Реализовать класс-шифровщик текстового файла с поддержкой различных алгоритмов шифрования. Возможные варианты шифрования: удаление всех гласных букв из текста, изменение букв текста на буквы, получаемые фиксированным сдвигом из алфавита (например, шифром буквы а будет являться буква д для сдвига 4 и т.д.), применение операции исключающее или с заданным ключом.

Паттерн "Стратегия" позволяет определить семейство алгоритмов, инкапсулировать каждый из них и делать их взаимозаменяемыми. Таким образом, можно изменять алгоритмы независимо от клиентов, которые их используют.

В данном случае, мы можем создать интерфейс "Шифровщик" (Encryptor), который определяет методы для шифрования текста. Затем мы можем создать классы, реализующие этот интерфейс, для каждого из предложенных алгоритмов шифрования (например, классы " VowelRemovalStrategy " и " LetterShiftStrategy ").

Работа программы:

```

Vowel Removal Encryption: Hll, Wrld!
Letter Shift Encryption: eeee
XOR Encryption: 0000

Process finished with exit code 0
|

```

Код программы:

```

interface EncryptionStrategy {
    String encrypt(String text);
}
class VowelRemovalStrategy implements EncryptionStrategy {
    @Override

```

```

        public String encrypt(String text) {
            return text.replaceAll("aeiouAEIOU", "");
        }
    }

    class LetterShiftStrategy implements EncryptionStrategy {
        private int shift;

        public LetterShiftStrategy(int shift) {
            this.shift = shift;
        }

        @Override
        public String encrypt(String text) {
            StringBuilder encryptedText = new StringBuilder();

            for (char c : text.toCharArray()) {
                if (Character.isLetter(c)) {
                    char encryptedChar = (char) (c + shift);
                    encryptedText.append(encryptedChar);
                } else {
                    encryptedText.append(c);
                }
            }

            return encryptedText.toString();
        }
    }

    class XorEncryptionStrategy implements EncryptionStrategy {
        private String key;

        public XorEncryptionStrategy(String key) {
            this.key = key;
        }

        @Override
        public String encrypt(String text) {
            StringBuilder encryptedText = new StringBuilder();

            for (int i = 0; i < text.length(); i++) {
                char c = text.charAt(i);
                char keyChar = key.charAt(i % key.length());
                char encryptedChar = (char) (c ^ keyChar);
                encryptedText.append(encryptedChar);
            }

            return encryptedText.toString();
        }
    }

    class TextFileEncryptor {
        private EncryptionStrategy encryptionStrategy;

        public void setEncryptionStrategy(EncryptionStrategy encryptionStrategy) {
            this.encryptionStrategy = encryptionStrategy;
        }

        public String encryptText(String text) {
            if (encryptionStrategy != null) {
                return encryptionStrategy.encrypt(text);
            } else {
                throw new IllegalStateException("Encryption strategy is not set.");
            }
        }
    }

    public class Lab6_3 {
        public static void main(String[] args) {
            TextFileEncryptor encryptor = new TextFileEncryptor();

            encryptor.setEncryptionStrategy(new VowelRemovalStrategy());
            String encryptedText1 = encryptor.encryptText("Hello, World!");
            System.out.println("Vowel Removal Encryption: " + encryptedText1);
        }
    }

```

```
        encryptor.setEncryptionStrategy(new LetterShiftStrategy(4));
        String encryptedText2 = encryptor.encryptText("aaaa");
        System.out.println("Letter Shift Encryption: " + encryptedText2);

        encryptor.setEncryptionStrategy(new XorEncryptionStrategy("secret"));
        String encryptedText3 = encryptor.encryptText("aaaa");
        System.out.println("XOR Encryption: " + encryptedText3);
    }
}
```

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