

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

Бондаренко К.А.

Проверил:

Крощенко А.А.

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

Ход работы

Вариант 5

Задание 1. Завод по производству смартфонов. Обеспечить создание нескольких различных моделей мобильных телефонов с заранее выбранными характеристиками.

Для данного задания воспользуемся порождающим паттерном Фабричный метод для создания объектов без указания их конкретных классов. Фабричный метод определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. В данном задании паттерн Фабричный метод подходит для создания различных моделей мобильных телефонов с заранее выбранными характеристиками.

Код программы:

Класс Смартфон:

```
public abstract class Smartphone {
    protected String model;
    protected int productionYear;
    protected int batteryCapacity;
    protected double screenSize;

    public Smartphone(String model, int productionYear, int batteryCapacity, double screenSize) {
        this.model = model;
        this.productionYear = productionYear;
        this.batteryCapacity = batteryCapacity;
        this.screenSize = screenSize;
    }

    public abstract void printInfo();
}
```

Класс смартфонов Bluemi:

```
public class Bluemi extends Smartphone {
    public Bluemi(String model, int productionYear, int batteryCapacity, double screenSize) {
        super(model, productionYear, batteryCapacity, screenSize);
    }

    @Override
    public void printInfo() {
        System.out.println("Смартфон Bluemi модели " + super.model
            + "\nГод выпуска: " + super.productionYear
            + "\nЁмкость батареи: " + super.batteryCapacity + " мА·ч")
    }
}
```

```

        + "\nРазмер экрана: " + super.screenSize + "\n");
    }
}

```

Класс смартфонов ISmartphone:

```

public class ISmartphone extends Smartphone {
    public ISmartphone(String model, int productionYear, int batteryCapacity, double screenSize) {
        super(model, productionYear, batteryCapacity, screenSize);
    }

    @Override
    public void printInfo() {
        System.out.println("Смартфон ISmartphone модели " + super.model
            + "\nГод выпуска: " + super.productionYear
            + "\nЁмкость батареи: " + super.batteryCapacity + " мА·ч"
            + "\nРазмер экрана: " + super.screenSize + "\n");
    }
}

```

Класс Фабрики создания смартфонов:

```

public abstract class SmartphoneCreator {
    public abstract Smartphone createSmartphone();
}

```

Класс Фабрики создания смартфонов Bluemi:

```

public class BluemiCreator extends SmartphoneCreator {
    @Override
    public Smartphone createSmartphone() { return new Bluemi("12 Pro", 2022, 4500, 6.5); }
}

```

Класс Фабрики создания смартфонов ISmartphone:

```

public class ISmartphoneCreator extends SmartphoneCreator {
    @Override
    public Smartphone createSmartphone() { return new ISmartphone("2", 2009, 3200, 4.5); }
}

```

Класс с методом main:

```

public class Main {
    public static void main(String[] args) {
        BluemiCreator bluemiCreator = new BluemiCreator();
        Smartphone bluemi = bluemiCreator.createSmartphone();
        bluemi.printInfo();

        System.out.println("");
        ISmartphoneCreator iSmartphoneCreator = new ISmartphoneCreator();
        Smartphone iSmartphone = iSmartphoneCreator.createSmartphone();
        iSmartphone.printInfo();
    }
}

```

Результаты работы программы:

```
Смартфон Bluemі модели 12 Pro
Год выпуска: 2022
Ёмкость батареи: 4500 мА·ч
Размер экрана: 6.5"

Смартфон ISmartphone модели 2
Год выпуска: 2009
Ёмкость батареи: 3200 мА·ч
Размер экрана: 4.5"
```

Задание 2. Проект «Электронный градусник». В проекте должен быть реализован класс, который дает возможность пользоваться аналоговым градусником так же, как и электронным. В классе «Аналоговый градусник» хранится высота ртутного столба и границы измерений (верхняя и нижняя).

Для данного задания воспользуемся структурным паттерном Адаптер. Паттерн Адаптер преобразует интерфейс класса к другому интерфейсу, на который рассчитан клиент. В данном случае мы можем создать адаптер, который преобразует интерфейс аналогового градусника в интерфейс электронного градусника.

Код программы:

Интерфейс Электронный градусник:

```
public interface ElectronicThermometer {
    double getTemperature();
}
```

Класс Аналоговый градусник:

```
public class AnalogThermometer {
    private final double lowerLimit;
    private final double upperLimit;
    private double mercuryHeight;
    private final double graduationScale;

    public AnalogThermometer(double lowerLimit, double upperLimit, double mercuryHeight, double graduationScale) {
        this.lowerLimit = lowerLimit;
        this.upperLimit = upperLimit;
        this.mercuryHeight = mercuryHeight;
        this.graduationScale = graduationScale;
    }

    public double getLowerLimit() { return lowerLimit; }

    public double getUpperLimit() { return upperLimit; }
```

```

public double getMercuryHeight() {
    return mercuryHeight;
}

public double getGraduationScale() {
    return graduationScale;
}

public void setMercuryHeight(double mercuryHeight) {
    if (mercuryHeight < 0) {
        mercuryHeight = lowerLimit;
        return;
    }

    if (lowerLimit + mercuryHeight / graduationScale > upperLimit) {
        mercuryHeight = lowerLimit;
        return;
    }

    this.mercuryHeight = mercuryHeight;
}
}

```

Класс Адаптер:

```

public class ThermometerAdapter implements ElectronicThermometer {
    private AnalogThermometer analogThermometer;
    public ThermometerAdapter(AnalogThermometer analogThermometer) {
        this.analogThermometer = analogThermometer;
    }

    @java.lang.Override
    public double getTemperature() {
        return analogThermometer.getLowerLimit() + analogThermometer.getMercuryHeight() /
analogThermometer.getGraduationScale();
    }
}

```

Класс с методом main:

```

public class Main {
    public static void main(String[] args) {
        ThermometerAdapter thermometerAdapter = new ThermometerAdapter(
            new AnalogThermometer(35.0, 42.0, 10.0, 2.0));
        System.out.println("Температура на градуснике: " + thermometerAdapter.getTemperature() + "
C°");
    }
}

```

Результаты работы программы:

```
Температура на градуснике: 40.0 C°
```

Задание 3. Проект «Банкомат». Предусмотреть выполнение основных операций (ввод пин-кода, снятие суммы, завершение работы) и наличие различных режимов работы (ожидание, аутентификация, выполнение операции, блокировка – если нет денег). Атрибуты: общая сумма денег в банкомате, ID.

Для данного задания воспользуемся поведенческим паттерном Состояние. Этот паттерн позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. В данном случае, банкомат может находиться в различных состояниях: ожидание, аутентификация, выполнение операции, блокировка.

Код программы:

Интерфейс Состояние банкомата:

```
interface ATMState {  
    void insertCard();  
    void ejectCard();  
    void enterPinCode(int pinCode);  
    void requestCash(int cashAmount);  
}
```

Класс Состояние ожидания:

```
public class ATMWaitingState implements ATMState {  
    private final ATM atm;  
  
    public ATMWaitingState(ATM atm) {  
        this.atm = atm;  
    }  
  
    @java.lang.Override  
    public void insertCard() {  
        System.out.println("Карта вставлена!");  
        atm.setCurrentState(new ATMAuthenticationState(atm));  
    }  
  
    @java.lang.Override  
    public void ejectCard() {  
        System.out.println("Невозможно извлечь карту - карта не вставлена!");  
    }  
  
    @java.lang.Override  
    public void enterPinCode(int pinCode) {  
        System.out.println("Карта не вставлена!");  
    }  
  
    @java.lang.Override  
    public void requestCash(int cashAmount) {  
        System.out.println("Вставьте сначала карту и введите пин-код!");  
    }  
}
```

Класс Состояние аутентификации:

```
public class ATMAuthenticationState implements ATMState {
    private final ATM atm;

    public ATMAuthenticationState(ATM atm) {
        this.atm = atm;
    }

    @java.lang.Override
    public void insertCard() {
        System.out.println("Карта уже вставлена!");
    }

    @java.lang.Override
    public void ejectCard() {
        System.out.println("Карта извлечена!");
        atm.setCurrentState(new ATMWaitingState(atm));
    }

    @java.lang.Override
    public void enterPinCode(int pinCode) {
        System.out.println("Пин-код введен!");
        atm.setCurrentState(new ATMOperationPerformingState(atm));
    }

    @java.lang.Override
    public void requestCash(int cashAmount) {
        System.out.println("Введите сначала пин-код!");
    }
}
```

Класс Состояние выполнения операции:

```
public class ATMOperationPerformingState implements ATMState {
    private final ATM atm;

    public ATMOperationPerformingState(ATM atm) {
        this.atm = atm;
    }

    @java.lang.Override
    public void insertCard() {
        System.out.println("Карта уже вставлена!");
    }

    @java.lang.Override
    public void ejectCard() {
        System.out.println("Карта извлечена!");
        atm.setCurrentState(new ATMWaitingState(atm));
    }

    @java.lang.Override
    public void enterPinCode(int pinCode) {
```

```

        System.out.println("Пин-код уже введен!");
    }

    @java.lang.Override
    public void requestCash(int cashAmount) {
        if (cashAmount > atm.getTotalCash()) {
            System.out.println("Недостаточно средств на карте!");
            atm.setCurrentState(new ATMBlockingState(atm));
            return;
        }
        int remainingCash = atm.getTotalCash() - cashAmount;
        atm.setTotalCash(remainingCash);
        System.out.println("Операция выполнена. Снято " + remainingCash);
    }
}

```

Класс Состояние блокировки:

```

public class ATMBlockingState implements ATMState {
    private final ATM atm;

    public ATMBlockingState(ATM atm) {
        this.atm = atm;
    }

    @java.lang.Override
    public void insertCard() {
        System.out.println("Банкомат заблокирован!");
    }

    @java.lang.Override
    public void ejectCard() {
        System.out.println("Банкомат разблокирован, карта извлечена!");
        atm.setCurrentState(new ATMWaitingState(atm));
    }

    @java.lang.Override
    public void enterPinCode(int pinCode) {
        System.out.println("Банкомат заблокирован!");
    }

    @java.lang.Override
    public void requestCash(int cashAmount) {
        System.out.println("Банкомат заблокирован!");
    }
}

```

Класс Банкомат:

```

public class ATM {
    private int totalCash;
    private int pinCode;
    private ATMState currentState;
}

```



```

public ATM(int totalCash) {
    this.totalCash = totalCash;
    currentState = new ATMWaitingState(this);
}

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

public void insertCard() {
    currentState.insertCard();
}

public void ejectCard() {
    currentState.ejectCard();
}

public void enterPinCode(int pinCode) {
    this.pinCode = pinCode;
    currentState.enterPinCode(pinCode);
}

public void requestCash(int cashAmount) {
    currentState.requestCash(cashAmount);
}

public void setTotalCash(int totalCash) { this.totalCash = totalCash; }

public int getTotalCash() { return totalCash; }
}

```

Класс с методом main:

```

public class Main {
    public static void main(String[] args) {
        ATM atm = new ATM(100);
        atm.insertCard();
        atm.enterPinCode(1234);
        atm.requestCash(50);
        atm.requestCash(60);
        atm.ejectCard();
    }
}

```

Результаты работы программы:

```

Карта вставлена!
Пин-код введен!
Операция выполнена. Снято 50
Недостаточно средств на карте!
Банкомат разблокирован, карта извлечена!

```

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