МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра интеллектуальных информационных технологий

# Отчет по лабораторной работе №6

По дисциплине «Современные платформы программирования»
Специальность ПО-8

Выполнил:
Липовик И.С.
студент группы ПО-8
Проверил:
ст. преп. кафедры ИИТ,
«__»_____2024 г.

Брест 2024

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

**Вариант 15**

**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.

**Smartphone.java**

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

    public Smartphone(String _model, String _OS, double _screenSize, int
_productionYear) {
        this.model = _model;
        this.OS = _OS;
        this.screenSize  =_screenSize;
        this.productionYear = _productionYear;
    }

    public abstract void printInfo();
}
```

**SamsungSmartphone.java**

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

    @Override
    public void printInfo() {
        System.out.println("Samsung smartphone"
                + "\nModel: " + super.model
                + "\nOS: " + super.OS
                + "\nScreen size: " + super.screenSize + "\""
                + "\nProduction year: " + super.productionYear);
    }
}
```

**AppleSmartphone.java**

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

    @Override
    public void printInfo() {
        System.out.println("Apple Smartphone"
                + "\nModel: " + super.model
```

```
                + "\nOS: " + super.OS
                + "\nScreen size: " + super.screenSize + "\""
                + "\nProduction year: " + super.productionYear);
    }
}
```

**SmartphoneCreator.java**

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

**SamsungSmartphoneCreator.java**

```
public class SamsungSmartphoneCreator extends SmartphoneCreator {
    @Override
    public Smartphone createSmartphone() {
        return new SamsungSmartphone("Galaxy Z flip 4", "Android", 7.5,
2023);
    }
}
```

**AppleSmartphone.java**

```
public class AppleSmartphoneCreator extends SmartphoneCreator {
    @Override
    public Smartphone createSmartphone() {
        return new AppleSmartphone("iPhone 14", "iOS", 6.5, 2022);
    }
}
```

**task1.java**

```
public class task1 {
    public static void main(String[] args) {
        AppleSmartphoneCreator appleSmartphoneCreator = new
AppleSmartphoneCreator();
        Smartphone appleSmartphone =
appleSmartphoneCreator.createSmartphone();
        SamsungSmartphoneCreator iSmartphoneCreator =  new
SamsungSmartphoneCreator();
        Smartphone iSmartphone = iSmartphoneCreator.createSmartphone();
        appleSmartphone.printInfo();
        System.out.println();
        iSmartphone.printInfo();
    }
}
```

**Пример**

```
D:\СПП\lab6\out\production\lab6>java task1
Apple Smartphone
Model: iPhone 14
OS: iOS
Screen size: 6.5"
Production year: 2022

Samsung smartphone
Model: Galaxy Z flip 4
OS: Android
Screen size: 7.5"
Production year: 2023
```

**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.

**Код программы**

**ElectronicThermometer.java**

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

**ThermometerAdapter.java**

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

    @Override
    public double getTemperature() {
        return analogThermometer.getLowerLimit() +
analogThermometer.getMercuryHeight() /
analogThermometer.getUnitOfMeasurement();
    }
}
```

**Task2.java**

```
public class task2 {
    public static void main(String[] args) {
        ThermometerAdapter thermometerAdapter = new ThermometerAdapter(
                new AnalogThermometer(35.0, 42.0, 10.0, 2.0));
        System.out.println("Temperature on the thermometer: " +
thermometerAdapter.getTemperature() + " C°");
    }
}
```

**Пример**

```
D:\СПП\lab6\out\production\lab6>java task2
Temperature on the thermometer: 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.

**ATM.java**

```java
public class ATM {
    private int totalCash;
    private int ID;
    private BankCard card;
    private ATMState currentState;

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

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

    public void insertCard(BankCard _card) {
        this.card = _card;
        currentState.insertCard();
    }

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

    public void enterPinCode(int pinCode) {
        if(!card.checkPinCode(pinCode)){
            System.out.println("Wrong PIN code!");
            return;
        }
        currentState.enterPinCode();
    }

    public void requestCash(int cashAmount) {
        if(!card.checkBalance(cashAmount)){
            System.out.println("Insufficient cash on card!");
            return;
        }
        if(cashAmount > totalCash){
            System.out.println("Insufficient cash in ATM!");
            return;
        }
        currentState.requestCash(cashAmount);
        if(totalCash==0){
            setCurrentState(new ATMBlockingState(this));
        }
    }

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

    public int getTotalCash() {
        return totalCash;
    }
```

```
    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }
}
```

**BankCard.java**

```
public class BankCard {
    private int pinCode;
    private int balance;
    BankCard(int _pinCode, int _balance) {
        this.pinCode = _pinCode;
        this.balance = _balance;
    }
    public Boolean checkPinCode(int _pinCode) {
        return this.pinCode == _pinCode;
    }
    public Boolean checkBalance(int _balance) {
        return this.balance>_balance;
    }
}
```

**ATMState.java**

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

**ATMWaitingState.java**

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

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

    @Override
    public void insertCard() {
        System.out.println("The card is already inserted!");
        atm.setCurrentState(new ATMAuthenticationState(atm));
    }

    @Override
    public void extractCard() {
        System.out.println("Cannot remove card - card not inserted!");
    }

    @Override
    public void enterPinCode() {
        System.out.println("Card not inserted!");
    }
```

```
    @Override
    public void requestCash(int cashAmount) {
        System.out.println("First insert the card and enter the PIN code!");
    }
}
```

**ATMAunthenticationState.java**

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

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

    @Override
    public void insertCard() {
        System.out.println("The card is already inserted!");
    }

    @Override
    public void extractCard() {
        System.out.println("Card extracted!");
        atm.setCurrentState(new ATMWaitingState(atm));
    }

    @Override
    public void enterPinCode() {
        System.out.println("PIN code entered!");
        atm.setCurrentState(new ATMOperationPerformingState(atm));
    }

    @Override
    public void requestCash(int cashAmount) {
        System.out.println("Enter your PIN code first!");
    }
}
```

**ATMOperationPerformingState.java**

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

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

    @Override
    public void insertCard() {
        System.out.println("The card is already inserted!");
    }

    @Override
    public void extractCard() {
        System.out.println("Card extracted!");
        atm.setCurrentState(new ATMWaitingState(atm));
    }
```

```
    @Override
    public void enterPinCode() {
        System.out.println("PIN code has already been entered!");
    }

    @Override
    public void requestCash(int cashAmount) {
        int remainingCash = atm.getTotalCash() - cashAmount;
        atm.setTotalCash(remainingCash);
        System.out.println("The operation is completed. Removed " +
remainingCash);
    }
}
```

**ATMBockingState.java**

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

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

    @Override
    public void insertCard() {
        System.out.println("ATM is blocked!");
    }

    @Override
    public void extractCard() {
        System.out.println("ATM is blocked, Card extracted!");
        atm.setCurrentState(new ATMWaitingState(atm));
    }

    @Override
    public void enterPinCode() {
        System.out.println("ATM is blocked!");
    }

    @Override
    public void requestCash(int cashAmount) {
        System.out.println("ATM is blocked!");
    }
}
```

**Task3.java**

```
public class task3 {
    public static void main(String[] args) {
        ATM atm = new ATM(100, 1);
        BankCard card = new BankCard(1234, 150);
        atm.insertCard(card);
        atm.enterPinCode(1234);
        atm.requestCash(50);
        atm.requestCash(60);
        atm.ejectCard();
```

```
    }
}
```

**Пример:**

```
D:\СПП\lab6\out\production\lab6>java task3
The card is already inserted!
PIN code entered!
The operation is completed. Removed 50
Insufficient cash in ATM!
Card extracted!
```

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