Пример 2: Программа "Payment"

Цель примера

Продемонстрировать процесс создания простейшей программы с использованием абстракции и полиморфизма.

Введение

Полиморфизм – это возможность идентично использовать объекты к одинаковым интерфейсом или родительском классом без информации о конкретном типе этого объекта. Поскольку Java является строго типизированным, при объявлении переменной необходимо указать тип данных, который будет храниться. Таким образом можно повысить безопасность кода, предотвратив потенциальные ошибки на этапе компиляции.

Абстракция позволяет описывать основные черты класса без конкретики, например описать действия, которые умеет выполнять класс, при этом не предоставляя конкретную реализацию. Таким образом, класс, который наследует абстрактный класс или реализует интерфейс, обязан предоставить конкретную логику работы метода.

Практическое руководство

Рассмотрим программу, которая представляет абстрактный платеж и его конкретные реализации.

Для этого разработаем интерфейс `Payment` с единственным методом `void process(int amount)` и создадим конкретные способы оплаты: банковский перевод и платеж через PayPal.

Шаг 1.

Создадим интерфейс `Payment`, который содержит единственный метод:

```
public interface Payment {

    void process(int amount);

}
```

Любой класс, который реализует этот интерфейс, обязан переопределить поведение этого метода.

ℹ️ **Информация**: Часто интерфейсы называют контрактами, поскольку они гарантируют наличие описанных методов в классе. Поэтому можно описывать сложную логику игнорируя конкретный класс, а используя полиморфизм, и не опасаться ошибок в момент компиляции или работы программы.

Шаг 2.

Создадим класс `BankPayment`, который реализует интерфейс, используя ключевое слово `implements`:

```
public class BankPayment implements Payment {

    private String senderBankAccount;
    private String receiverBankAccount;

    public BankPayment(String senderBankAccount, String receiverBankAccount) {
        this.senderBankAccount = senderBankAccount;
        this.receiverBankAccount = receiverBankAccount;
    }

    @Override
    public void process(int amount) {
        System.out.println("Processing Card Payment:");
        System.out.println("Transferring '" + amount + "€' from '" + senderBankAccount + "'
to '" + receiverBankAccount + "'");
    }

}
```

Обратите внимание, что класс имеет свой собственный набор свойств и конструктор. В рамках примера, для выполнения платежа через банк нужно знать исходящий и входящий банковские счета. Таким образом, метод `process()` обязывает указать только сумму платежа, а специфичная логика перечисления реализуется самим классом.

Шаг 3.

Создадим класс `PaypalPayment`, который реализует интерфейс:

```
public class PaypalPayment implements Payment {

    private String senderEmail;
    private String receiverEmail;

    public PaypalPayment(String senderEmail, String receiverEmail) {
        this.senderEmail = senderEmail;
        this.receiverEmail = receiverEmail;
    }

    @Override
    public void process(int amount) {
        System.out.println("Processing Paypal Payment:");
        System.out.println("Transferring '" + amount + "€' from PayPal '" + senderEmail + "'
to PayPal '" + receiverEmail + "'");
    }

}
```

В отличие от класса `BankPayment`, для выполнения платежа нужно знать исходящий и входящий почтовые адреса клиентов. Сигнатура метода по прежнему требует только сумму платежа.

Шаг 4.

Продемонстрируем работу в классе `PaymentDemo`. Для этого создадим разные платежи, где в качестве типа данных объявим `Payment`. Во вспомогательный метод `executePayment()` передадим сумму платежа и метод, которым мы выполнить платеж. Обратите внимание, что сам метод не знает, какой механизм платежа будет использован.

```
public class PaymentDemo {

    public static void main(String[] args) {

        Payment bankPayment = new BankPayment("LV123454321BANK", "LV567898765BANK");
        Payment paypalPayment = new PaypalPayment("sender@mail.lv", "receiver@mail.lv");

        executePayment(bankPayment, 15);

        System.out.println();

        executePayment(paypalPayment, 20);

    }

    public static void executePayment(Payment payment, int amount) {
        System.out.println("Executing payment:");
        payment.process(amount);
    }

}
```

Результат работы программы:

> Executing payment: Processing Card Payment: Transferring '15€' from 'LV123454321BANK' to 'LV567898765BANK'
>
> Executing payment: Processing Paypal Payment: Transferring '20€' from PayPal 'sender@mail.lv' to PayPal 'receiver@mail.lv'

⚠️ **Важно**: При объявлении платежа `Payment bankPayment = new BankPayment(...)` можно указывать полный тип: `BankPayment bankPayment = new BankPayment(...)`, при этом в метод `executePayment)` можно также передать это платеж в качестве параметра, поскольку банковский платеж реализует интерфейс `Payment`.

Рекомендации:

- Запустить программу и сравнить результаты;
- Изменить тип переменных `Payment` на `BankPayment` и `PaypalPayment` и сравнить результаты;
- Добавить собственную реализацию механизма платежа, например через кредитную карту;