Základy objektovo orientovaného programovania

Java: OOP jednoducho

Triedy a objekty, zapuzdrenie, dedičnosť, polymorfizmus, abstrakcia a kompozícia

1. Trieda a objekt

Trieda je opis toho, aké údaje a aké činnosti môžu mať objekty určitého typu.
Objekt je konkrétna inštancia triedy.
Napríklad môžeme vytvoriť triedu `Cat`:

```
public class Cat {
    String name;

    public void meow() {
        System.out.println("Meow!");
    }
}
```

Potom môžeme vytvoriť objekt:

```
Cat cat = new Cat();

cat.name = "Murka";
cat.meow();
```

Tu `Cat` je trieda a `cat` je objekt tejto triedy.

Hlavná myšlienka

Objekt zvyčajne spája:

- údaje — jeho **stav**;
- metódy — jeho **správanie**.

2. Zapuzdrenie

Zapuzdrenie znamená, že vnútorný stav objektu je chránený pred nesprávnymi zmenami.
Pozrime sa na hráča, ktorý má zdravie:

```
public class Player {

    private int health = 100;

    public void takeDamage(int damage) {
        if (damage > 0) {
            health = Math.max(0, health - damage);
        }
    }

    public int getHealth() {
        return health;
    }
}
```

Pole `health` je označené ako `private`. Iný kód preto nemôže ľubovoľne nastaviť jeho hodnotu.
Stav sa mení pomocou zrozumiteľnej metódy:

```
player.takeDamage(20);
```

Takýto kód však nie je povolený:

```
player.health = -500; // Chyba
```

Zapuzdrenie pomáha objektu udržiavať správny stav.

Dôležité

Zapuzdrenie neznamená iba použitie `private`.
Hlavnou myšlienkou je dovoliť meniť stav iba bezpečným a zmysluplným spôsobom.

3. Dedičnosť

Dedičnosť umožňuje vytvoriť novú triedu na základe už existujúcej triedy. Zvyčajne vyjadruje vzťah:

„je typom“.

Napríklad pes a mačka sú zvieratá.

```
public class Animal {

    public void sound() {
        System.out.println("Some sound");
    }
}
```

Trieda `Dog` dedí z triedy `Animal`:

```
public class Dog extends Animal {

    @Override
    public void sound() {
        System.out.println("Woof!");
    }
}
```

Trieda `Cat` môže vytvoriť vlastnú verziu tej istej metódy:

```
public class Cat extends Animal {

    @Override
    public void sound() {
        System.out.println("Meow!");
    }
}
```

Kľúčové slovo `extends` znamená, že jedna trieda dedí z druhej.

4. Polymorfizmus

Polymorfizmus umožňuje pracovať s rôznymi objektmi prostredníctvom spoločného typu.
Napríklad:

```
Animal animal = new Dog();

animal.sound();
```

Premenná má typ `Animal`, ale skutočným objektom je `Dog`.
Preto sa zavolá metóda z triedy `Dog`:

```
Woof!
```

Môžeme pracovať aj s viacerými zvieratami:

```
Animal[] animals = {
    new Dog(),
    new Cat()
};

for (Animal animal : animals) {
    animal.sound();
}
```

Výsledok:

```
Woof!
Meow!
```

Rovnaké volanie

```
animal.sound();
```

môže vykonať rôzne činnosti podľa toho, aký objekt sa v premennej skutočne nachádza.

5. Abstrakcia

Abstrakcia nám pomáha sústrediť sa na to, **čo** objekt dokáže robiť, bez toho, aby sme museli poznať všetky podrobnosti o tom, **ako** to robí.
V Jave sa na tento účel často používajú rozhrania.
Napríklad:

```
public interface Switchable {

    void turnOn();

    void turnOff();
}
```

Rozhranie nám hovorí, že:

- objekt možno zapnúť;
- objekt možno vypnúť.

Nevysvetľuje však, ako presne sa tieto činnosti vykonajú.
Napríklad lampa môže toto rozhranie implementovať:

```
public class Lamp implements Switchable {

    public void turnOn() {
        System.out.println("Lamp is on");
    }

    public void turnOff() {
        System.out.println("Lamp is off");
    }
}
```

Kľúčové slovo `implements` znamená, že trieda implementuje rozhranie.

6. Kompozícia

Kompozícia znamená, že jeden objekt obsahuje alebo používa iný objekt.
Vyjadruje vzťah:

„má“.

Napríklad auto má motor.

```
public class Engine {

    public void start() {
        System.out.println("Engine started");
    }
}
```

Teraz môžeme motor použiť vo vnútri auta:

```
public class Car {

    private final Engine engine =
        new Engine();

    public void start() {
        engine.start();
    }
}
```

Trieda `Car` nie je motor.
Auto **má** motor a používa ho.

7. Dedičnosť alebo kompozícia?

Pomôcť môže jednoduchá otázka.
Je jeden objekt typom druhého objektu?
Vtedy môže byť vhodná dedičnosť.
Napríklad:

Dog is an Animal.

Pes je zviera.
Má jeden objekt iný objekt?
Vtedy je zvyčajne vhodná kompozícia.
Napríklad:

Car has an Engine.

Auto má motor.

Dôležité

Dedičnosť nepoužívajte iba preto, aby ste získali prístup k už napísanému kódu.
Najprv skontrolujte, či medzi triedami naozaj existuje vzťah „je typom“.

8. Prepisovanie metódy

Prepisovanie metódy znamená, že podtrieda poskytne vlastnú implementáciu metódy, ktorá už existuje v rodičovskej triede.
Napríklad trieda `Animal` obsahuje:

```
public void sound() {
    System.out.println("Some sound");
}
```

Trieda `Dog` môže túto metódu prepísať:

```
@Override
public void sound() {
    System.out.println("Woof!");
}
```

Anotácia

```
@Override
```

pomáha kompilátoru skontrolovať, že skutočne prepisujeme už existujúcu metódu.
Prepisovanie metód úzko súvisí s polymorfizmom.

9. Preťažovanie metódy

Preťažovanie znamená, že existuje viacero metód s rovnakým názvom, ale s rôznymi parametrami.
Napríklad:

```
public void print(String text) {
    System.out.println(text);
}

public void print(int number) {
    System.out.println(number);
}
```

Teraz môžeme napísať:

```
print("Hello");
print(25);
```

V prvom prípade sa zavolá metóda s parametrom `String`.
V druhom prípade sa zavolá metóda s parametrom `int`.

Nezamieňajte si ich

Prepisovanie — nová implementácia zdedenej metódy.
Preťažovanie — rovnaký názov metódy, ale rôzne parametre.

10. Ako vyzerá dobrá trieda?

Dobrá trieda zvyčajne:

- má jednu jasnú úlohu;
- uchováva iba údaje, ktoré potrebuje;
- nedovoľuje náhodne poškodiť svoj stav;
- používa zrozumiteľné názvy polí a metód;
- poskytuje operácie, ktoré zodpovedajú významu objektu.

Napríklad pre bankový účet je zrozumiteľnejšia metóda:

```
account.withdraw(100);
```

ako priama zmena zostatku:

```
account.balance = -100;
```

Prvý zápis opisuje činnosť „vybrať peniaze“ a umožňuje triede skontrolovať, či je operácia povolená.

11. Štyri hlavné princípy OOP

Zapuzdrenie
Chráni vnútorný stav objektu a poskytuje bezpečné spôsoby práce s ním.
Dedičnosť
Umožňuje vytvárať nové typy na základe existujúcich typov, ak medzi nimi skutočne existuje vzťah „je typom“.
Polymorfizmus
Umožňuje používať rôzne objekty prostredníctvom spoločného typu.
Abstrakcia
Sústreďuje sa na dôležité vlastnosti objektu a skrýva nepotrebné detaily implementácie.

Hlavné myšlienky OOP

Trieda opisuje objekty.
Objekt uchováva stav a vykonáva činnosti.
Zapuzdrenie chráni stav.
Dedičnosť vytvára vzťahy medzi príbuznými typmi.
Polymorfizmus umožňuje spoločnému typu pracovať s rôznymi implementáciami.
Abstrakcia ukazuje dôležité možnosti objektu a skrýva detaily.
Kompozícia umožňuje vytvárať zložitejšie objekty z jednoduchších objektov.

Táto pomôcka je určená na prvé zoznámenie s objektovo orientovaným programovaním. Príklady sú zámerne jednoduché, aby ukázali hlavné myšlienky bez zbytočných detailov.