8. - Dědičnost, method overriding, function overloading

Dědičnost

Dědičnost umožňuje třídám využívám proměnné a metody jiné třídy a zabraňuje zbytečnému duplikaci kódu do jiných tříd. Dalo by se to také popsat jako vytváření třídy využitím jiné třídy. Třída, která poskytuje svoje proměnné a své metody se nazývá rodič, třída, která naopak získává, se nazývá potomek.

Typický příklad může být třída Animal, od které dědí například Dog. Třída Animal má název a věk, třída Dog má společný ještě navíc proměnnou plemeno.

```java
public class Anima Close
    private String name;
    private int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void makeSound() {
        System.out.println("The animal makes a sound.");
    }
}
```

```java
public class Dog extends Animal {
    private String breed;

    public Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }

    public String getBreed() {
        return breed;
    }
    @Override
    public void makeSound() {
        System.out.println("The dog barks.");
    }
}
```

Kdybychom chtěli, aby při zavolání Dog.makeSound() se vyvolala metoda z rodiče, tudíž výpis „This animal makes a sound.“, tak by stačilo metodu u třídy Dog odebrat.

Jak to ale funguje vevnitř?

Dědičnost v Javě je implementována pomocí mechanismu tzv. virtuálních metod, což znamená, že když voláme metodu z objektu odděděné třídy, JVM vyhledává definici metody nejprve v odděděné třídě a pokud ji tam nenajde, vyhledá ji v jejím rodiči, a tak dále až do té nejvyšší třídy v hierarchii. Rodiče pozná pomocí extends.

Method Overriding

Method Overriding je, když v odděděné třídě (V našem případě v třídě **Dog**) vytvoříme funkci se stejným názvem, který je už v rodičovi (**makeSound**) a poté upravíme její vnitřek. Defaultně, kdybychom totiž u třídy **Dog** neměli žádnou metodu makeSound, jak jsem již výše zmínil, JVM by našel v rodičovi metodu makeSound a tu by využil.

Pokud ale chceme, aby pes štěkal místo toho, aby vypisoval **„This animal makes a sound“,** tak si vytvoříme ve třídě **Dog** metodu **makeSound** a ta overridne metodu ze třídy Animal.

Nad metodu v odděděné třídě, v tomto případě **makeSound**, se píše @Override, což je anotace a značí, že tato metoda něco přepisuje.

Function overloading

Function overloading nastává v případě, že potřebujeme mít více funkcí se stejným jménem v jedné třídě, ale potřebujeme, aby měli různé argumenty nebo různý návratový typ. Příkladem může být například třída kalkulačka, která může mít dvě stejně pojmenované funkce, jedna bude přičítat celá čísla, druhá bude přičítat například desetinná (int / float)

Function overloading nám umožňuje vytvářet více verzí určité funkce. Představme si například, že chceme vytvořit funkci pro výpočet obvodu geometrických útvarů. Můžeme vytvořit funkci s názvem "obvod", která bude mít různé verze pro různé typy geometrických útvarů (kruh, čtverec, trojúhelník atd.). Každá verze funkce bude mít jiné vstupní parametry, ale budou mít stejný návratový typ. Pokud budeme chtít zjistit obvod kruhu, zavoláme funkci **obvod()** s jedním parametrem (poloměr kruhu), pokud chceme zjistit obvod čtverce, zavoláme funkci **obvod()** s dvěma parametry (délka strany čtverce), atd. Tímto způsobem můžeme vytvořit jednoduše čitelný a udržovatelný kód.