Metoda (objektově orientované programování)

Jump to navigationJump to search

Metody v objektově orientovaném programování (OOP) jsou podobné funkcím, které se používají v programování. V OOP jsou v podstatě také funkcemi, které mohou pracovat s daty třídy nebo objektu. Z vnějšku jsou data neviditelná – nepřístupná, jelikož jsou zapouzdřena v objektu a nelze je volat přímo. Metody určené k tomu, aby s daty objektu mohly pracovat i jiné objekty, nazýváme rozhraním objektu.

Obsah

[skrýt]

Typy metod[editovat | editovat zdroj]

Statické metody[editovat | editovat zdroj]

Jsou součástí třídy, ale lze je použít aniž by byla vytvářena instance třídy. Chceme-li označit danou metodu jako statickou, použijeme klíčové slovo *static*.

Konečné (finální) metody[editovat | editovat zdroj]

Metody se mohou v odvozených třídách překrývat, ale existují případy, kdy chceme mít jistotu, že danou metodu nebude možné změnit. Slouží k tomu konečné metody, které nemohou být „přepsány“ v žádné z odvozených tříd. Chceme-li označit danou metodu jako konečnou, použijeme klíčové slovo *final*.

Abstraktní (virtuální) metody[editovat | editovat zdroj]

Při návrhu se lze setkat s případy, kdy chceme nechat implementaci určitých metod až na potomky. Jako příklad lze uvést třídu *Obrazec*, která definuje obecný geometrický útvar a u níž víme, že potomci budou mít stejné metody (např. obvod a obsah), ale jejich implementace bude různá. Pokud tedy označíme metodu jako *abstraktní* říkáme tím, že tuto metodu implementuje její potomek. Tyto metody se většinou označují klíčovým slovem *abstract* nebo *virtual*.

Výše uvedený příklad v Javě:

```
//definování obecného Obrazce
abstract class Obrazec {
     //každý potomek této třídy např. Čtverec, Obdélník bude implementovat
níže uvedené metody
     public abstract double obvod();
     public abstract double obsah();
}
```

```
//třída definující Čtverec a implementující dvě abstraktní metody předka
class Ctverec extends Obrazec {
    public double obvod(double a){
        return double (4*a);
    }
    public double obsah(double a){
        return double (a*a);
    }
}

//třída definující Obdélník a implementující dvě abstraktní metody předka
class Obdelnik extends Obrazec {
    public double obvod(double a, double b){
        return double (2*(a+b));
    }
    public double obsah(double a, double b){
        return double (a*b);
    }
}
```

Speciální metody[editovat | editovat zdroj]

Sem patří konstruktor, destruktor a tzv. *„gettery“* a *„settery“*. Poslední dvě zmiňované metody neslouží k ničemu jinému, než k nastavování a získávání hodnot daných vlastností. Není vhodné vlastnosti tříd definovat jako *veřejné*, mohlo by např. dojít k nechtěné změně jejich hodnot, ale jako *chráněné* nebo *soukromé* a k těmto vlastnostem pak přistupovat pomocí zmíněných *getterů* a *setterů*. Jejich názvy by měly začínat slovem *get_* nebo *set_*.

Modifikátory přístupu[editovat | editovat zdroj]

Většina běžně používaných objektově orientovaných programovacích jazyků má tři modifikátory přístupu:

- **public** - lze je volat odkudkoli
- **protected** - lze je volat pouze z metod stejné či odvozené třídy
- **private** - lze je volat pouze z metod téže třídy

Přetěžování[editovat | editovat zdroj]

Umožňuje objektům volání jedné metody se stejným jménem, ale s jinou implementací. Provádí se to tak, že se deklaruje více metod se stejným názvem, které se mohou lišit různým počtem, typem argumentů popř. jejich pořadím.
Ukázka v Javě:

```
// Třída pro práci se soubory
public class Soubor{
    public static void otevri(Binary binary){
        Console.WriteLine("Otevírám binární soubor.");
```

```
        //implementace otevření binárního souboru
    }
    public static void otevri(Text text){
        Console.WriteLine("Otevírám textový soubor.");
        //implementace otevření textového souboru
    }

    public static void prectiRadek(Binary binary){
        Console.WriteLine("Čtu řádek z binárního souboru.");
        //implementace čtení řádku z binárního souboru
    }
    public static void prectiRadek(Text text){
        Console.WriteLine("Čtu řádek z textového souboru.");
        //implementace čtení řádku z textového souboru
    }

    public static void zavri(Binary binary){
        Console.WriteLine("Zavírám binární soubor.");
        //implementace zavření binárního souboru
    }
    public static void zavri(Text text){
        Console.WriteLine("Zavírám textový soubor.");
        //implementace zavření textového souboru
    }
}
```

Volání metod[editovat | editovat zdroj]

Rozlišujeme volání uvnitř třídy a volání metody určitého objektu. V prvním případě se běžně používá pouze její název, ve druhém případě je nejdříve uveden název objektu a pak název volané metody.
Ukázka volání metod v Javě:

```
//definování třídy
public class Trida {
    double Metoda1 (double a){
        return a;
    }
    double Metoda2 (double b){
        //volání metody uvnitř třídy
        double prom = Metoda1(b);
        return prom;
    }
}
```

```
Trida instance=new Trida();
//volání metody objektu
instance.Metoda1(5);
```

Volání metod může být v některých programovacích jazycích jiné viz PHP, kde se volají následovně:

```
class MojeTrida{
    //statická metoda vypisující chyby
    static function vypisChybu($zprava_chyby, $cislo_chyby){
        echo "Nastala chyba číslo: $cislo_chyby";
        echo "Popis chyby: $zprava_chyby";
    }
    public function deleni(a, b){
        if((a || b) == 0){
            //volání lokální statické metody uvnitř třídy pomocí self
            self::vypisChybu("Nelze dělit nulou!","400");
        }else {
            return (a/b);
        }
    }
    public function vypocet(a, b){
        $vysledek = 0;
        //volání lokální metody uvnitř třídy pomocí $this
        $vysledek = $this->deleni(a,b)+(a*b);
        return $vysledek;
    }
}
//volání metody objektu
$trida = new MojeTrida();
$trida->vypis_chybu(4,5);
```

Ukázky vytvoření metod[editovat | editovat zdroj]

Java[editovat | editovat zdroj]

```
//Ukázka statických metod
public class Ctverec extends Obrazec {
    //deklarace statické metody pro výpočet obvodu
    static double obvod (double a){
        return 4*a;
    }
    //deklarace statické metody pro výpočet obsahu
    static double obsah (double a){
```

```
        return a*a;
    }
}

//Ukázka finálních metod
class Ctverec extends Obrazec {
    final double obvod (double a){
        return 4*a;
    }
    final double obsah (double a){
        return a*a;
    }
}
```