# Objektově orientované programování

V současné době jeden z nejčastěji používaných přístupů, existuje ve velké většině programovacích jazyků.

Řešení problému s využitím OOP je víceméně **analogií k postupu, kterým by tento problém řešil člověk.**

U OOP je kladen důraz na **znuvupoužitelnost**. Přístup nalézá inspiraci v průmyslové revoluci – vynález základních komponent, které budeme dále využívat (např. když stavíme dům, nebudeme si pálit cihly a soustružit šroubky, prostě je již máme hotové).

Poskládání programu z komponent je výhodnější a levnější. Můžeme mu věřit, je otestovaný (o komponentách se ví, že fungují, jsou otestovány a udržovány). Pokud je někde chyba, stačí ji opravit **na jednom místě.**

**Využívá se konstrukce známé z procedurálního programování** (funkce, proměnné atd.), které skládá do složitějších celků.

Co umožňuje OOP programátorovi:

* abstrakci (zobecnění) pro modelování a řešení problémů,
* znovupoužitelný kód,
* kontrola přístupu k datům,
* minimalizace samovolných / nezamyšlených zásahů do kódu,

OOP představuje přístup, jak správně navrhnout strukturu programu (tj. jeho architekturu) tak, aby byl program funkční a udržovatelný.

**Princip fungování objektově orientovaného programování:**

## Třída a objekt:

V objektově orientovaných jazycích se programy skládají dohromady z prvků zvaných třídy. Každá třída je umístěna ve svém vlastním souboru, který se jmenuje stejně jako ona. Třída může mít svoje proměnné – **atributy** – a svoje **metody**. Je to obecný předpis toho, jaké vlastnosti a chování mají objekty do této třídy patřící. **Objektem** se pak nazývají konkrétní prvky – **instance** nějaké třídy.

**Příklad třídy a jejich objektů:**

Uvažujte **třídu Pes**. Každý pes má čtyři nohy, nějakou barvu, nějaký věk, umí štěkat a běhat. Konkrétním **objektem** náležejícím do třídy **Pes** by pak byl třeba pes **Punťa**, který má čtyři nohy, hnědou barvu, je mu 7 let a samozřejmě umí štěkat a běhat.

Je zřejmě jasné, že objekt patřící do třídy zdaleka nemusí být jen jeden, na světě jistě není pouze jeden pes Punťa, konkrétních mazlíčků, které bychom označili jako psa, je mnohonásobně více.

**Rozdělení na proměnné a metody:**

Uvedené vlastnosti, jako jsou například barva, či věk, by byly převedeny **na atributy** třídy Pes neboli na její proměnné. Vlastnost počet nohou by navíc mohla být konstanta, typický pes má vždy čtyři nohy. Chování a schopnosti jako běhání nebo štěkání by byly vyjádřeny **metodami** této třídy.

**Deklarace třídy**

Každý program napsaný v Javě musí obsahovat alespoň jednu třídu. Třída se zapisuje pomocí klíčových slov ***public class Název třídy*** a složené závorky. Veškerý obsah třídy, její proměnné i metody, musí být umístěn uvnitř těchto složených závorek.

Deklarace třídy Pes:

public class Pes { // deklarace třídy Pes

public final int POČET\_NOHOU = 4; // konstanta počet nohou

public String barva; // text s názvem barvy

public int vek; // číslo udávající věk

public void stekej[] { // metoda štěkej

}

public void behej[] { // metoda běhej

}

}

Konstanta POČET\_NOHOU může mít uvedenou svou hodnotu, protože bude pro všechny objekty této třídy stejná (všichni psi mají čtyři nohy), ostatní proměnné mají uveden pouze modifikátor přístupu (public – viz. níže), datový typ a název.

**Důležité:** Názvy tříd podle konvence **začínají velkým počátečním písmenem**. Pokud se skládají z více slov, každé další slovo také začíná velkým písmenem a **neodděluje se mezerou**. Mezi další dobré zvyky patří pojmenovávání tříd podstatnými jmény (Pes, LoveckyPes)

### Modifikátory

Vlastnosti a viditelnost proměnných, metod i tříd můžeme nastavovat pomocí modifikátorů. Jazyk Java rozlišuje tři modifikátory přístupu, které určují, z jakých míst programu je modifikovaný prvek viditelný. Modifikátor se zapisuje do deklarace prvku na první místo.

Modifikátor ***public***: označuje se jako veřejný a umožňuje přístup danému prvku z libovolného místa v programu. Důležité je, že daný prvek je také možno upravovat, což nemusí být vždy žádoucí. Každý program musí obsahovat alespoň jednu veřejnou třídu, což přímo vyplývá z nutnosti přístupu k hlavní metodě programu main. Metoda main musí být uvedena ve veřejné třídě, v jiném případě ohlásí překladač chybu.

Modifikátor ***private***: přistup pouze z třídy, kde byl člen deklarován.

Modifikátor ***protected***: přistup ze třídy, kde byl člen deklarován + přístup z odvozených tříd (tzv. potomků, viz níže).

### Modifikátor static

Jako striktně objektově orientovaný jazyk používá Java při své práci především objekty. Někdy však může být vhodné některé prvky programu používat bez nutnosti vytvoření rodičovského objektu. Při použití s modifikátorem přístupu zapisujeme klíčové slovo static za něj na druhé místo v deklaraci prvku.

Public static int pocet;

**Statické proměnné** existují nezávisle na vytvořených objektech a náleží pouze třídě, ve které jsou deklarovány. Statické proměnné existují pouze jedenkrát pro celou třídu. Nepopisují vlastnost objektů třídy, ale třídy jako takové. Příkladem pro třídu Pes může být počet psů uchovávaných v daném programu. Jejich výhodou je, že k nim lze přistupovat ze statických metod tříd, aniž by musel být vytvořen objekt, který je instancí dané třídy. Pozn.: lokální proměnné deklarované uvnitř metod třídy nemohou být vytvořeny jako statické.

**Statické metody** stejně jako proměnné existují nezávisle na objektech třídy a lze je volat bez vytvoření objektu. Nemohou přistupovat k proměnným náležejícím objektům, pouze ke statickým proměnným své třídy, lze je tedy volat i v případě, že neexistuje žádný objekt dané třídy.

public static metoda() {}

**Hlavní metoda main je deklarována jako statická právě proto, že je nutné ji spustit ještě před tím, než se v programu vytvoří jakýkoli objekt, hned po načtení hlavní třídy.**

Metody, které lze v rámci třídy volat nad jejím objektem se pak nazývají **členské metody**.

### Modifikátor final

Pokud chcete v programu zabránit změně nějakého prvku, použijete k tomu modifikátor final. Je možné ho použít v souvislosti s proměnnými, metodami i třídami. Nejjednodušší použití je při **deklaraci konstanty**. Pokud tento modifikátor přidáte do deklarace proměnné, **nebude možné jí změnit hodnotu, která by jednou přiřazena**. Klíčové slovo final se zapisuje za případný modifikátor přístupu i za případně uvedený modifikátor static.

private final double CISLO\_PI = 3,14

public static final int CHYBA = -1

Když se v programu pokusíte změnit hodnotu proměnné deklarované jako final, ohlásí překladač chybu. Použití modifikátoru final u metod **zabraňuje jejich přepsání**. Podobně při použití v souvislosti se třídou bude **znemožněno její dědění**. Třída deklarovaná jako final nemůže mít žádné své podtřídy v rámci celého programu.

## Metody

Metody slouží k tomu, aby zadaným způsobem upravovaly získaná data a prováděly další činnosti. Jak již bylo vysvětleno, existují statické metody (nezávislé na objektech) a členské metody (volány nad objektem).

### Deklarace metody

Před použitím metody potřebuje překladač Javy některé informace, které jsou uvedeny v její deklaraci. U metod lze použít **modifikátory přístupu**, dále modifikátor **static** i modifikátor **final**. Důležitou informací pro překladač je **datový typ návratové hodnoty**. Ten specifikuje, jakého typu má být výsledná hodnota vrácená touto funkcí, a je pečlivě kontrolován. Každá funkce má také svůj **název** a **seznam parametrů** v kulatých závorkách.

public static void main(String[] args) {}

Metoda main je veřejně dostupná (**public**) a lze ji spustit i bez vytvoření objektu (**static**). Klíčové slovo **void** označuje prázdnou návratovou hodnotu, metoda vykonává nějaké příkazy, ale nic nevrací. Název metody je **main** a v kulatých závorkách je její parametr.

**Důležité:** I pojmenování metod má své konvence. Název začíná **malým písmenem** a teprve počáteční písmena případných dalších slov názvu začínají písmenem velkým. Je také dobrým zvykem volit jako jména metod **slovesa** (např. secti, vypis).

### Parametry metody

Metody mohou pracovat s proměnnými svých objektů. Pokud potřebují ke své práci další data (atributy), mohou jim být předána ve formě argumentů. Argumenty se zapisují za název metody do kulatých závorek ve formě:

(typ nazevParametru, typ nazevParametru2)

Těmto argumentům se při používání uvnitř metody říká **formální parametry**. Metoda také může mít prázdný seznam argumentů. V tom případě pracuje pouze s atributy svého objektu a kulaté závorky při její deklaraci jsou prázdné. Počet parametrů není omezen, nesmí se ovšem vyskytovat dva parametry se stejným názvem.

### Návratová hodnota

Metoda může sama skončit po provedení všech příkazů uvnitř bloku, pokud je její návratová hodnota definována jako typ void. V tom případě metoda pouze vykonává nějakou činnost, ale nevrací žádný výsledek. Ukončit provádění metody je možné i dříve pomocí příkazu **return**. Samotný příkaz return vrátí běh programu na první příkaz uvedený za místem, ze kterého byla metoda volána.

public **void** projdi(int a, int b) {

if (a<b) {

**return**;

}

for (int i = a; i >= b; i++) {

suma += i; // použití atributu

}

}

Často metody uvnitř svého těla počítají nějakou hodnotu, která je **potřeba pro další průběh programu**. V tom případě se v deklaraci metody uvede její datový typ a za návratový příkaz return se doplní jméno proměnné obsahující potřebnou hodnotu.

public int secti(int a, int b) {

int soucet = a + b;

**return soucet**;

}

Příkazem return není nutné vracet jen hodnotu proměnné, na její místo lze zapsat i výraz a vrácena bude jeho výsledná hodnota.

public int secti(int a, int b) {

**return a + b;**

}

### Přetěžování metod

V rámci jedné třídy je možné definovat **více metod se stejným názvem** i stejným návratovým typem, pokud se jejich **formální parametry liší**. Deklarace takové metody se nazývá přetížení. Parametry se mohou odlišovat buď počtem, nebo svým datovým typem. Návratový typ se může také lišit, ovšem samotná jeho odlišnost k přetížení nestačí.

public void secti() { //základní metoda sečti s prázdným návratovým typem

soucet = a + b //využívá atributy objektu

}

public void secti(int a, int b) { // přetížená metoda se stejným návratovým typem

soucet = a + b;

}

public int secti(int a, int b, int c) { // přetížená metoda s jiným návratovým typem

return a + b + c; // má parametr navíc oproti předchozí

}

****Užitečnost přetěžování metod je dobře vidět například na funkcích ze třídy Math, která poskytuje matematické funkce a operace. Pokud třeba hledáte maximum ze dvou čísel, stačí použít metodu max(a, b) pro všechny číselné typy. Vždy se jmenuje stejně. Kdyby tato možnost neexistovala, museli bychom si pamatovat několik metod, například maxInt, maxDouble a podobně, pro každý typ zvlášť.

## Objekty

Třída sama o sobě není úplně užitečná, dokud nevytvoříme nějaké její objekty. S těmito objekty potom program dále pracuje, volá jejich metody a upravuje jejich atributy.

K vytváření nových objektů slouží klíčové slovo **new**. Při deklaraci objektu se jako jeho datový typ uvede název třídy, ze které bude objekt vytvořen, před názvem samotného objektu. Za operátorem přiřazení (=) pak místo hodnoty zapíšeme právě klíčové slovo new, mezeru a znovu název třídy s následujícími kulatými závorkami:

Pes Punta = new Pes();

Mezi název třídy a kulaté závorky na konci příkazu se nikdy nepíše mezera. Zápis Třída() představuje volání speciální metody objektu.

**Konstruktor:**

Touto speciální metodou je takzvaný konstruktor, který inicializuje objekt při jeho vytvoření. Pokud není ve třídě uveden, použije se obecný konstruktor, který nemá žádné argumenty a nepředává svým atributům žádné hodnoty, pouze vytvoří nový objekt. Předávat hodnotu atributům objektu až po jeho vytvoření je mnohdy nepraktické, proto je vhodné konstruktor ve třídě uvést. Jako tvůrce objektu musí být konstruktor vždy veřejná metoda, která má stejný název jako třída, ve které je uveden. Nemá také žádný návratový typ, protože vždy vrací odkaz na vytvořený objekt.

Příklad:

public class Pes {

**Public Pes()** {}

}

Tento konstruktor zatím práci moc neusnadňuje, protože se chová stejně jako ten obecný. Jako běžným metodám i konstruktoru lze předat argumenty, typicky takové, jejichž hodnoty se uloží do atributů objektu.

public class Pes {

public static final int POČET\_NOHOU = 4;

public String barva;

public int vek;

**public Pes(String b, int v)** { //konstruktor číslo 1

barva = b;

vek = v;

}

}

I konstruktor lze přetěžovat jako běžné metody, můžete tedy uvést několik jeho variant v závislosti na počtu atributů, které chcete inicializovat při vytvoření objektu. Do třídy Pes by tedy bylo možné připsat i konstruktory:

public Pes(String b) { // konstruktor číslo 2

barva = b;

}

public Pes(int v) { // konstruktor číslo 3

vek = v;

}

Svůj vytvořený konstruktor pak použijete při vytváření nového objektu tak, že uvedete jeho formální parametry jako argumenty do kulatých závorek při použití klíčového slova new. Java si podle jejich počtu a typu správně vybere, který konstruktor má použít.

Pes hnedy = new Pes(“hneda“); // použije konstruktor 2

Pes stary = new Pes(15); // použije konstruktor 3

Pes mladyZeleny = new Pes(“zelena“, 1); // použije konstruktor 1

## Klíčové slovo this

Ačkoli konstruktory uvedené v předchozí části splnily svůj účel, je jasné že **b** a **v** nejsou příliš vhodné názvy parametrů. Po pár měsících je možné, že už nebudeme vědět, co přesně tyto parametry představují. Vhodné názvy se pro tyto parametry přímo nabízejí, očividně předáváme barvu a věk psa, ovšem tyto názvy už používají atributy třídy.

Parametry metody (včetně konstruktoru) představují lokální proměnné, jejichž platnost je omezena na příslušný bok složených závorek. Lokální proměnné mají ovšem schopnost zastínit globální proměnné (proměnné objektu) se stejným názvem. Řešením je použití klíčového slova **this**.

Každá metoda při svém zavolání dostane kromě uvedených argumentů navíc i odkaz na objekt, nad kterým byla volána. Tento odkaz je přístupný právě s pomocí klíčového slova this, a umožňuje tedy rozlišovat mezi atributem objektu a parametrem metody. K atributům se přistupuje pomocí tečkové notace – za slovo this se tedy napíše tečka a název atributu. Původní konstruktor tedy lze upravit:

public Pes(String barva, int vek) {

this.barva = barva;

this.vek = vek;

}

Atributům barva a vek objektu, na který odkazuje klíčové slovo this, budou přiděleny hodnoty lokálních proměnných barva a vek, předaných jako argumenty konstruktoru.