Objektovo orientované programovanie

(triedy, objekty, členy objektov, členy tried, konštruktory, preťažovanie metód, referenčný typ, argumenty metódy, prístupové práva, zapuzdrenie, kompozícia, agregácia, konštanty)

3. prednáška

Vladislav Novák
FEI STU v Bratislave
30.9.2014
(úprava po prednáške na str. 16)
(úprava 17.10. na str.8)
(oprava tabuľky na str. 11)

Obsah

Deklarácia triedy

minimálna deklarácia triedy:

```
class NazovTriedy {
     // deklarácia atribútov, konštruktorov a metód
}
```

- atribúty – ukladanie stavu – implementácia pomocou premenných
- konštruktory – automaticky vykonávajú inicializáciu nových objektov
- metódy – operácie nad triedou, alebo nad jej inštanciami (objektmi) – implementácia pomocou funkcii

Názov triedy (konvencia)

- jednoslovné názvy - prvé písmeno je veľké, ostatné písmena sú malé
- viacslovné názvy – podobne ako jednoslovné názvy, ale prvé písmeno každého slova je veľké

príklady:

```
Bicykel
Auto
HorskyBicykel
OsobnéAuto
```

Členské premenné (atribúty)

```
modifikátorPrístup typPremennej názovPremennej;
```

Modifikátor prístupu určuje, ktoré triedy majú prístup k danej členskej premennej.

Ako typ členskej premennej sa môže použiť niektorý zo základných typov (int, double, char, atď), alebo referenčný typ (trieda, rozhranie, pole).

Názov premennej - identifikácia

Členské metódy

```
modifikátorPrístupu
návratovýTyp
názovMetody
(typParametra nazovParametra, typParametra2 nazovParametra2)
zoznam výnimiek
{
     //definícia implementácie
}
```

Signatúra metódy

Signatúra metódy slúži na identifikáciu metódy (vzájomné rozlíšenie metód)
Signatúra metódy sa skladá z názvu a typov parametrov.
príklad:
metóda:

```
public double vypocitaj(double sirka, double vyska, int pocet)
```

signatúra:

```
vypocitaj(double, double, int)
```

Pomenovanie metód (konvencia)

- prvé slovo názvu metódy sa píše malými písmenami, ďalšie slová sa začínajú veľkým písmenom za ktorým nasledujú malé písmená
- prvé slovo názvu je sloveso

príklady:

```
bez
bezRychlo
získajPozadie
získajVysledneUdaje
jePrazdne
```

Preťažovanie metód (overloading methods)

Preťažovanie metód – vytvorenie viacerých metód s rovnakým názvom, ale odlišným typom alebo počtom vstupných parametrov (s rôznou signatúrov).
Pri vzájomnom rozlišovaní metód nerozhoduje typ návratovej hodnoty, takže nemožno v jednej triede vytvoriť dve metódy s rovnakou signatúrou, ale rôznymi typmi návratových hodnôt.
Pri volaní preťaženej metódy sa vhodná verzia metódy vyberie podľa typu a počtu parametrov uvedených pri jej volaní.

Príklad: používanie preťaženej metódy `System.out.println(`*`parametre`*`)`:
Definícia v triede `PrintStream`:

```
..... class PrintStream ..... {
    ..............
    public void println() {.....}
    public void println(boolean x) {.....}
    public void println(char x) {.....}
    public void println(int x) {.....}
    public void println(long x) {.....}
    public void println(float x) {.....}
    public void println(double x) {.....}
    public void println(char[] x) {.....}
    public void println(String x) {.....}
    public void println(Object x) {.....}
    ............
}
```

Volanie metódy:

```
int cislo = 10;
String retazec = "ahoj";
System.out.println(cislo);  //volanie println(int x)
System.out.println(retazec);//volanie println(String x)
```

Príklad 2: rôzne počty a typy parametrov:

```
public class KresliacePlatno {
    public void vytlac(String retazec) {..... }
    public void vytlac(String retazec, int poziciaX, int poziciaY){}
    public void vytlac(int cislo) {.....}
    public void vytlac(int cislo, int poziciaX, int poziciaY) {...}
}
```

Konštruktory

Konštruktor je procedúra, ktorá sa automatický volá pri vytváraní objektu. Slúži pre inicializáciu objektu.
Deklarácia konštruktora je podobná ako deklarácia bežnej metódy, ale neobsahuje návratový typ (konštruktor nevracia hodnotu) a názov je rovnaký ako názov triedy.

Príklad deklarácie triedy (konštruktor, členské premenné, členské metódy) a jej použitie

```
public class Obdlznik {
    private String nazov;
    private int sirka;
    private int vyska;

    public Obdlznik(String pocNazov,
                    int pocSirka, int pocVyska){
        this.nazov = pocNazov;
        this.sirka = pocSirka;
        this.vyska = pocVyska;
    }

    public void tlacInfo() {
        System.out.println(this.nazov + ": "
                           + this.sirka + ", "  + this.vyska);
    }

    public static void main(String[] args){
        Obdlznik obdl = new Obdlznik("moj obdlznik",11,22);
        obdl.tlacInfo(); //vytlaci: moj obdlznik: 11, 22
    }
}
```

Operátor `new`

Kľúčové slovo `new` je operátor jazyka Java. Tento operátor vyhradí v pamäti miesto pre objekt (inštanciu), zavolá konštruktor objektu, ktorý je uvedený za slovom `new` a vráti referenciu na novovytvorený objekt.

V predchádzajúcom príklade príkaz

```
Obdlznik obdl = new Obdlznik("moj obdlznik",11,22);
```

vytvorí v pamäti miesto pre inštanciu triedy `Obdlznik`, zavolá uvedený konštruktor a referenciu na novovytvorenú inštanciu nakopíruje do deklarovanej premennej `obdl`.

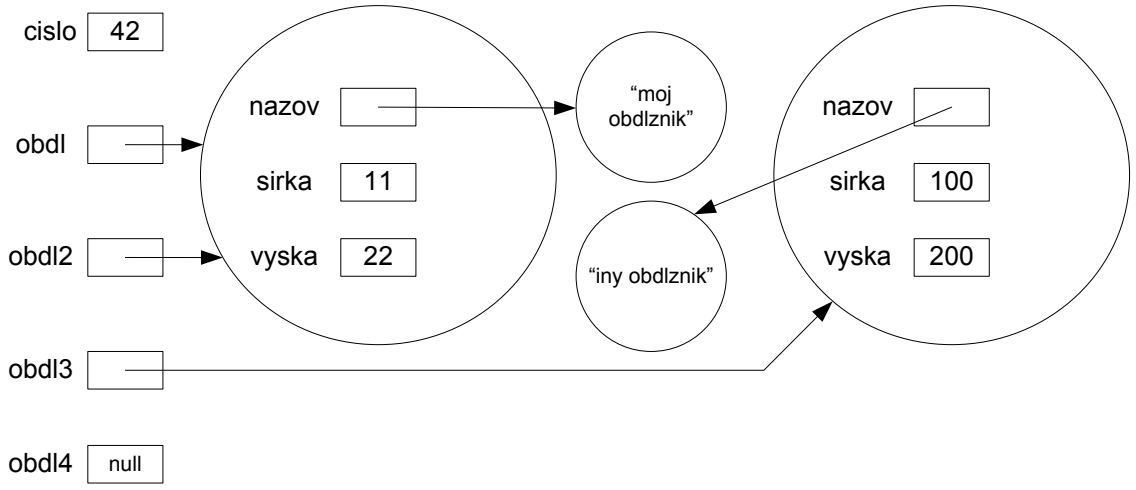
Kľúčové slovo `this` v hore uvedenom príklade označuje objekt, nad ktorým sa práve vykonáva metóda.

Referenčný typ premennej

Ak je typom premennej základný typ (int, double, ...) , potom premenná obsahuje priamo hodnotu daného typu.
Ak je typom premennej trieda (alebo iný referenčný typ), potom premenná môže obsahovať iba referenciu na objekt (neobsahuje samotný objekt). Ak premenná neobsahuje referenciu na objekt, tak je hodnota referencie `null`. Referenciu na jeden objekt môže obsahovať viacero premenných.

príklad (nadväzuje na predchádzajúci príklad):

```
int cislo = 42;
Obdlznik obdl = new Obdlznik("moj obdlznik", 11, 22);
Obdlznik obdl2 = obdl;
Obdlznik obdl3 = new Obdlznik(„iny obdlznik", 100, 200)
Obdlznik obdl4 = null;
```



Prístup k atribútom a metódam objektu

Používa sa operátor **.** (bodka) v tvare:

- z vnútra objetku:

```
this.nazovPremennej //this je možné vynechať
```

alebo

```
this.nazovMetódy(parametre) //this je možné vynechať
```

- mimo objektu:

```
referenciaNaObjekt.nazovPremennej
```

alebo

```
referenciaNaObjekt.nazovMetódy(parametre)
```

Poznámka:
Operátor **.** (bodka) sa využíva aj na prístup k metódam tried o ktorých je napísané ďalej.

Implicitný konštruktor (default constructor)

Ak v triede nie je explicitne definovaný žiadny konštruktor, prekladač automaticky vytvorí implicitný konštruktor bez parametrov. Tento konštruktor má rovnaké prístupové práva ako trieda.

príklad (implicitný konštruktor):

```
public class Obdlznik {
    private String nazov;
    private int sirka;
    private int vyska;

    public void tlacInfo() {
        System.out.println(this.nazov + ": "
                           + this.sirka + ", "  + this.vyska);
    }

    public static void main(String[] args){
        Obdlznik o1 = new Obdlznik();//vykona sa implicit.konstr.
        o1.tlacInfo(); //vytlaci: null, 0, 0
    }
}
```

Kompilátor priradil neinicializovaným členským premenným nulové hodnoty.

Poznámka: Implicitný konštruktor vyvolá konštruktor nadtriedy bez parametrov. Ak nadtrieda neobsahuje takýto konštruktor, tak vznikne chyba pri preklade (toto súvisí s dedičnosťou, ktorou sa budeme zaoberať neskôr).

Preťažovanie konštruktorov

Java umožňuje preťažovať aj konštruktory. Mechanizmus je podobný ako pri obyčajných metódach.

príklad:

```
public class Obdlznik {
    private String nazov;
    private int sirka;
    private int vyska;

    public Obdlznik() {
        this(0,0);
    }

    public Obdlznik(int pocSirka, int pocVyska){
        this("",pocSirka,pocVyska);
    }

    public Obdlznik(String pocNazov,
                    int pocSirka, int pocVyska){
        this.nazov = pocNazov;
        this.sirka = pocSirka;
        this.vyska = pocVyska;
    }

    public void tlacInfo() {
        System.out.println(this.nazov + ": "
                           + this.sirka + ", "  + this.vyska);
    }

    public static void main(String[] args){
        Obdlznik o1 = new Obdlznik();
        o1.tlacInfo(); //vytlaci: : 0, 0

        Obdlznik o2 = new Obdlznik(10,20);
        o2.tlacInfo(); //vytlaci: : 10, 20

        Obdlznik o3 = new Obdlznik("treti",11,22);
        o3.tlacInfo(); //vytlaci: treti: 11, 22
    }
}
```

Kľúčové slovo `this` je v tomto príklade použité aj na vyvolanie konštruktora (ďalšie podrobnosti neskôr).

Predávanie vstupných argumentov

Argumenty základného typu (napr.: `int`, `double`) aj referenčného typu (napr. objekty) sa predávajú hodnotou.

Parameter základného typu:
Do metódy sa predáva kópia hodnoty parametra. Zmeny hodnoty parametra vykonané v metóde sa preto mimo metódy neprejavia. To isté platí aj pre konštruktory.

Parameter referenčného typu:
Do metódy sa predáva kópia referencie na objekt. To znamená, že zmeny hodnôt v objekte zostávajú aj po ukončení metódy. Zmeny samotnej referencie vykonané v metóde sa mimo metódy neprejavia. To isté platí aj pre konštruktory.

príklad:

```
class Udaje {
    public int atribut;
}

public class PredavanieParametrov {

    public static void vytlacInkrement(int x) {
        x++;
        System.out.println(x);
    }

    private static void zmenUdaj(Udaje udaj) {
        udaj.atribut = 20;
        udaj = new Udaje();
        udaj.atribut = 30;
    }

    public static void main(String[] args) {
        int x = 0;
        vytlacInkrement(x);    //vytlaci: 1
        System.out.println(x); //vytlaci: 0

        Udaje udaj = new Udaje();
        udaj.atribut = 10;
        System.out.println(udaj.atribut); //vytlaci: 10
        zmenUdaj(udaj);
        System.out.println(udaj.atribut); //vytlaci: 20
    }
}
```

Automatická správa pamäti (garbage collector)

Nové objekty sa vytvárajú pomocou operátora `new`. Odstraňovanie nepotrebných objektov je v Jave automatické (menšia pracnosť a náchylnosť voči chybám).
Automatická správa pamäti pravidelne uvoľňuje nepoužívané objekty z pamäte, v intervaloch nezávislých na programátorovi.

Objekt možno odstrániť ak žiadna premenná neobsahuje referenciu na objekt.

Neskôr si uvedieme:
metóda finalize() – metóda volaná pri odstraňovaní objektu
System.runFinalization() //vynútenie spustenie finalizéru

príklady:

```
String ret1 = new String(”oblak”);
String ret2 = new String(”ahoj”);
String ret3 = new String(”stolicka”);
String ret4 = ret3;

ret1 = null;                //”oblak” môže byť odstránený
ret2 = new String(”cau”); //”ahoj” môže byť odstránený
ret3 = null;                //na ”stolicka” ešte obsahuje referenciu ret4
ret4 = null;                //”stolicka” môže byť odstránený

void nazovMetody() {
    ret = new String(”mesto”);
} //ak na ”mesto” už neodkazuje žiadna premenná, tak ho po skončení metódy možno odstrániť
```

Automatická správa pamäti nezaručuje odstránenie všetkých nepotrebných objektov. Ak existuje premenná obsahujúca referenciu na nepotrebný objekt, tak algoritmus automatickej správy môže, ale nemusí identifikovať objekt ako objekt, ktorý treba odstrániť.

Preto môže byť v niektorých prípadoch dôležité nulovať referencie na nepotrebné objekty.

```
Obdlznik obdlznik = new Obdlznik(/*parametre*/); //vytvorenie objektu

//používanie objektu

obdlznik = null; //už objekt nepotrebujeme, automatická správa pamäte ho môže odstrániť
```

Použitie kľúčového slova this

Prístup k členskej premennej objektu, alebo členskej metóde objektu

V rámci členskej metódy, alebo konštruktora predstavuje kľúčové slovo `this` odkaz na objekt nad ktorým sa metóda, alebo konštruktor práve vykonáva. Pomocou `this` možno vo vnútri členskej metódy, alebo konštruktora odkazovať na ľubovoľnú členskú premennú, alebo členskú metódu. V tomto prípade možno slovo `this` väčšinou vynechať.

Poznámka:
Kľúčové slovo `this` nemôžeme použiť pre prístup ku statickým členom (patriacim triede, nie jej inštanciám), ktoré sú popísané ďalej.

príklady

```
public class Bod{
    private int x;
    private int y;

    //Tri verzie nastavenia:

    public void nastav1(int noveX, int noveY) {
        this.x = noveX;
        this.y = noveY;
    }

    //to iste ako nastav1, ale môžeme vynechať this
    public void nastav2(int noveX, int noveY) {
        x = noveX;
        y = noveY;
    }

    //Parametre metódy majú rovnaký názov ako členské premenné.
    //V metóde bude x označovať vstupný parameter x, nie členskú premennú x (podobne y).
    //Preto pre označenie členskej premennej musíme použiť kľúčové slovo this.
    public void nastav3(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Volanie konštruktora

V konštruktore možno pomocou kľúčového slova `this` vyvolať iný konštruktor tej istej triedy.
Ak konštruktor obsahuje volanie iného konštruktora, musí byť toto volanie uvedené na jeho prvom riadku.

príklad:

```
public class Bod {
    private String nazov;
    private int x;
    private int y;

    public Bod() {
        this("",0,0); //volanie konštruktora musí byť prvé
        //tu môže byť ďalší kód
    }

    public Bod(String nazov) {
        this(nazov,0,0); //volanie konštruktora musí byť prvé
        //tu môže byť ďalší kód
    }

    public Bod(int x, int y) {
        this("",x,y); //volanie konštruktora musí byť prvé
        //tu môže byť ďalší kód
    }

    public Bod(String nazov, int x, int y) {
        this.nazov = nazov;
        this.x = x;
        this.y = y;
    }
}
```

Riadenie prístupových práv (viditeľnosti) pomocou modifikátorov

Trieda, členská premenná, metóda, alebo konštruktor je v danej oblasti:

- viditeľná = prístupná = môžeme s ňou v danej oblasti priamo pracovať

alebo

- neviditeľná = neprístupná = nemôžeme s ňou v danej oblasti pracovať

Táto časť súvisí s dedičnosťou a balíkmi, ktorými sa budeme zaoberať neskôr.
Balík je zoskupenie tried.

Riadenie prístupových práv k triede

Ak je pred deklaráciu triedy uvedený modifikátor `public`, tak je daná trieda viditeľná z ľubovoľného miesta (môžeme ju používať na ľubovoľnom mieste).

Ak pred deklaráciou triedy nie je uvedený žiadny modifikátor, tak je trieda viditeľná iba v rámci vlastného balíka. Toto východzie nastavenie bez modifikátora sa označuje ako *package-private*.

príklady:

```
//trieda viditeľná všade
public class Trieda1 {
    //.....
}

//trieda viditeľná iba v rámci balíku (package-private)
class Trieda2 {
    //.....
}
```

Riadenie prístupových práv k členom inštancii a tried

modifikátor `public` – člen inštancie/triedy je prístupný na ľubovoľnom mieste
modifikátor `protected` – člen inštancie/triedy je dostupný vo vlastnej triede, v podtriedach (v ľubovoľnom balíku) a v celom svojom balíku
bez modifikátora (*package-private*) – člen inštancie/triedy je dostupný v celom svojom balíku
modifikátor `private` – člen inštancie/triedy je prístupný iba vo vlastnej triede

Tabuľka prístupových práv:

modifikátor	v rámci triedy	v inej triede toho istého balíka	v podtriede, ktorá je v inom balíku	v inej triede, iného balíka
`public`	ANO	ANO	ANO	ANO
`protected`	ANO	ANO	ANO	NIE
bez modifikátoru	ANO	ANO	NIE	NIE
`private`	ANO	NIE	NIE	NIE

Doporučenia:
Treba nastavovať čo najnižšie prístupové práva .
Členské premenné je väčšinou vhodné deklarovať pomocou modifikátoru `private`.

Hlavnou výhodou nastavenia čo najnižších prístupových práv je jednoduchšia možnosť meniť implementáciu v prípade potreby (zapuzdrenie, minimalizácia previazania triedy s okolím).

Zapuzdrenie (encapsulation)

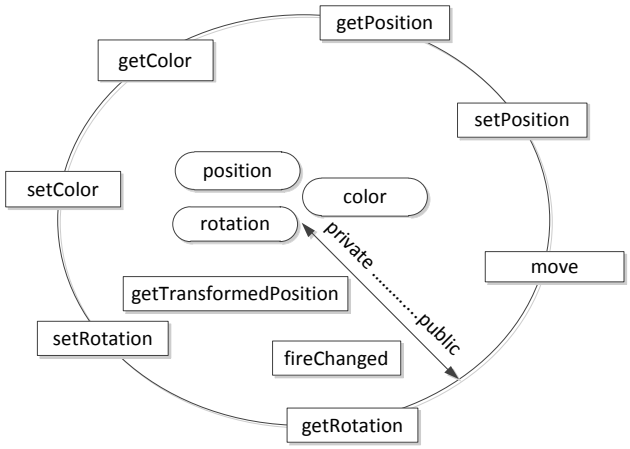
Zapuzdrenie je zoskupenie súvisiacich ideí do jednej jednotky, na ktorú sa možno odkazovať jediným názvom.

Príkladom môže byť podprogram (procedúra, funkcia) ktorá zapuzdruje inštrukcie.
- Na podprogram sa možno odkazovať jediným názvom.
- Šetrí nie len pamäť počítača, ale aj ľudskú námahu. Uľahčuje tvorbu zložitých programov, pretože predstavuje koncepčný balík, ktorý možno považovať za samostatnú ideu (na určitej úrovni), ktorú možno potom takto používať.

Objektovo orientované zapuzdrenie – zabalenie atribútov (predstavujúcich stav objektu) a operácii (metód) do jednej jednotky (do jedného typu objektu). Stav je prístupný a meniteľný iba cez rozhranie poskytované zapuzdrením[1].

Zapuzdrenie:
- pohľad zvnútra (obsahuje podrobnosti o objekte)
- pohľad zvonku (tvorené rozhraním, neobsahuje podrobnosti o objekte)

Vonkajší pohľad väčšinou obsahuje iba niektoré operácie, ktoré tvoria rozhranie objektu, cez ktoré môžeme s objektom pracovať. V jave ich nastavujeme pomocou prístupových práv.



Obrázok 1 Trieda Rectangle - zapuzdrenie stavu a operácii nad obdĺžnikom

Zapuzdrenie zahrňuje aj skrývanie informácii a implementácii.

Skrývanie informácii znamená, že informácie v objekte nie sú z vonkajšieho pohľadu prístupné. Používa sa pre informácie, ktoré nie sú dôležité pri pohľade zvonku.

Skrývanie implementácie znamená, že podrobnosti implementácie nie sú z vonku prístupné. Príkladom skrytia implementácie môže byť trieda `Obdĺžnik`, ktorá poskytuje metódu `dajObsah()`, vracajúcu obsah obdĺžnika. Používateľ tejto triedy ale napríklad nevie kedy dochádza k výpočtu obsahu (obsah sa môže vypočítať pri zmene dĺžok strán a uložiť od atribútu, alebo sa jeho hodnota neukladá do atribútu, ale sa vypočíta vždy pri volaní metódy `dajObsah()`).

[1] Pod rozhraním nie je myslená programová konštrukcia `interface` (napr. v jazyku java), ale časti objektu viditeľné z vonkajšieho pohľadu.

Výhodou dobrého zapuzdrenia je flexibilita pri úpravách. Pozorovateľ zvonku vie čo objekt dokáže, ale nevie ako. Ak robíme zmeny v skrytej implementácii, tak tieto zmeny majú, minimálny dopad na zvyšok systému.

Súčasťou dobrého zapuzdrenia je povolenie prístupu k atribútom iba cez metódy (väčšia bezpečnosť a ľahšia modifikovateľnosť implementácie).

Úrovne zapuzdrenia
- 0. úroveň – kód bez zapuzdrenia
- 1. úroveň – podprogram (procedúra, funkcia)
- 2. úroveň – trieda (objekt)
- ďalšie úrovne – balíky (len niektoré triedy sú dostupné mimo balíka), komponenty

Asociácia celku a časti

Dve najpoužívanejšie asociácie celku a časti sú kompozícia a agregácia.

Kompozícia (zloženie)
- Zložený objekt (celok, kompozícia) neexistuje bez svojich častí (zložiek, komponentov).
- V každom okamihu môže byť akýkoľvek komponent súčasťou iba jednej kompozície.
- Ak zanikne celok, tak zaniknú aj všetky jeho časti. Časti nemusia vzniknúť pri vzniku celku.
- Kompozícia je väčšinou heterometrická (komponenty sú väčšinou rôznych typov). Napríklad lietadlo sa sklad z trupu a krídiel.

Agregácia (zoskupenie)
- Zoskupený objekt (agregácia) môže potenciálne existovať bez svojich častí (tvoriacich/ konštitučných objektov). Napr. oddelenie môže existovať aj bez zamestnancov (táto vlastnosť nemusí byť vždy užitočná).
- Jeden objekt môže byť súčasťou viacerých zoskupení.
- Agregácia má tendenciu byť homeometrická (tvoriace objekty sú väčšinou toho istého typu). Napríklad les je agregáciou stromov, stádo zoskupením oviec.

Agregácia je špeciálna forma asociácie. Kompozícia je špeciálne forma agregácie (platia pre ňu prísnejšie podmienky ako pre agregáciu).

Členské premenné a metódy triedy (statické premenné a metódy)

Členské premenné a metódy objektu (inštancie)
- každý objekt má svoje vlastné premenné
- metódy sa vykonávajú nad objektom

Členské premenné a metódy triedy
- každá premenná existuje práve raz (patrí triede, resp. všetkým objektom). Takéto premenné existujú aj v prípade že nie je vytvorený žiadny objekt.
- metódy sa vykonávajú nad triedou. Pre zavolanie metódy nie je potrebné aby existoval objekt.

Členské premenné a metódy triedy sa označujú kľúčovým slovom `static`. Preto sa nazývajú *statické*.

Na statické premenné a metódy triedy sa môžeme odkazovať pomocou triedy, alebo pomocou objektu. Odkazovanie pomocou objektu sa neodporúča, pretože zo zápisu nie je jasné, že je to premenná resp. metóda triedy.
príklady:
```
Trieda.premenná //odporúčaný spôsob
objekt.premenná //neodporúča sa
Trieda.metoda()//odporúčaný spôsob
objekt.metoda()//neodporúča sa
```

Obmedzenia v metódach triedy:
- Metódy objektu môžu priamo pristupovať k premenným a metódam triedy.
- Metódy triedy môžu priamo pristupovať k premenný a metódam triedy.
- Metódy triedy nemôžu priamo pristupovať k premenný a metódam objektu, pretože sa metóda vykonáva nad triedou. Pre prístup k premennej alebo metóde objektu, musia použiť referenciu na objekt. Nemožno použiť slovo `this` na prístup k premenným a metódam objektu.

Napríklad trieda `java.lang.Math` obsahuje premenné (konštanty π, e) a metódy pre matematické výpočty (napr.: sínus, mocnina, maximum, logaritmus). Jej premenné a metódy sú statické vďaka čomu nemusíme vytvárať jej inštancie pri výpočtoch.
príklad:
```
int a = Math.max(2,4);
double b = Math.sin(0.1);
double c = 2 * Math.PI * polomer;
```

Ďalší príklad na statické premenné a metódy bude uvedený nižšie.

Konštanty

Pre definíciu konštanty slúži kľúčové slovo `final`. Presnejšie, označuje že hodnotu možno nastaviť iba raz.
príklad:

```
public void metoda() {
    final int MAX = 10; //priradenie pri deklarácii
    final int MIN;
    MIN = -10; //priradenie mimo deklarácie
    //MIN = -20; <- CHYBA konštanta už má definovanú hodnotu
}
```

Konštanta sa často definuje kombináciou modifikátorov `static` a `final`.
príklad:

```
public class Kalkulacka {
     private static final double PI = 3.14159265358979323846;
     //.....
}
```

Ak je konštanta základného typu, alebo reťazec a jej hodnota je v dobe prekladu známa, nahradí prekladač na všetkých miestach v kóde názov konštanty jej hodnotou.

Pomenovanie konštánt (konvencia):
Názov konštanty sa skladá z veľkých písmen. Ak je názov zložený z viacerých slov, tak sú slová oddelené podčiarkovníkom _
príklady:

```
final int MAXIMUM;          //jednoslovný názov
final int VIACSLOVNY_NAZOV; //viacslovný názov
```

Príklad na statické členy a konštanty

```
public class Vyrobok {
    private static int pocetVyrobkov = 0;
    private final int ID;

    public static int dajPocetVsetkych() {
        return pocetVyrobkov;
    }

    public Vyrobok() {
        pocetVyrobkov++;
        ID = pocetVyrobkov;
    }

    public int dajIdVyrobku() {
        return ID;
    }


    public static void dalsiaMetoda(Vyrobok v) {
        //int a = ID;       <- CHYBA
        //int b = this.ID; <- CHYBA
        int c = v.ID;     //<- OK
    }


    public static void main(String[] args) {
        Vyrobok v1 = new Vyrobok();
        Vyrobok v2 = new Vyrobok();

         System.out.println(Vyrobok.dajPocetVsetkych ()); //2
        System.out.println(v1.dajIdVyrobku()); //1
        System.out.println(v2.dajIdVyrobku()); //2

        Vyrobok v3 = new Vyrobok();
        System.out.println(v3.dajIdVyrobku()); //3
        System.out.println(Vyrobok.dajPocetVsetkych ()); //3
    }
}
```

Inicializácia členských (statických aj nestatických) premenných

Členské premenné (statické aj nestatické) možno inicializovať využitím rôznych konštrukcii.

Inicializácia inštančných premenných:	Inicializácia statických premenných:
- konštruktor	
- inicializácia pri deklarácii	- inicializácia pri deklarácii
- inicializačné bloky	- statický inicializačné bloky
- finálne metódy[2]	- súkromné statické metódy

Kompilátor skopíruje nestatické inicializačné bloky do všetkých konštruktorov.
Inicializačných blokov (statických aj nestatických) môže byť viacero.
Statické inicializačné bloky sa vykonajú v poradí v ktorom sú napísané.

Príklady inicializácie inštančných premenných:

```
public class InicializaciaKonstruktorom {
    private int instancnaPremenna1;
    private int instancnaPremenna2;

    public InicializaciaKonstruktorom() {
        this.instancnaPremenna1 = 100;
        this.instancnaPremenna2 = 200;
    }
}
```

```
public class InicializiaciaPriDeklaracii {
    private int instancnaPremenna1 = 100;
    private int instancnaPremenna2 = 200;
}
```

```
public class InicializaciaInicializacnyBlok {
    private int instancnaPremenna1;
    private int instancnaPremenna2;

    { //inicializacny blok pre premenne instancie
        instancnaPremenna1 = 100;
        instancnaPremenna2 = 200;
    }
}
```

```
public class InicializaciaFinalneMetody {
    private int instancnaPremenna1 = finalnaInicializacnaMetoda1();
    private int instancnaPremenna2 = finalnaInicializacnaMetoda2();

    protected final int finalnaInicializacnaMetoda1() {
        return 100;
    }

    protected final int finalnaInicializacnaMetoda2() {
        return 200;
    }
}
```

[2] Finálne metódy sú metódy, ktoré nemožno v podtriedach prekryť (meniť implementáciu). Finálnymi metódami sa budeme zaoberať neskôr.

Príklady inicializácie statických premenných:

```
public class InicializaciaPriDeklaracii {
    private static int statickaPremenna1 = 100;
    private static int statickaPremenna2 = 200;
}
```

```
public class InicializaciaStatickyInicializacnyBlok {
    private static int statickaPremenna1;
    private static int statickaPremenna2;

    static { //staticky inicializacny blok
     statickaPremenna1 = 100;
     statickaPremenna2 = 200;
    }
}
```

```
public class InicializaciaSukromneStatickeMetody {
    private static int statickaPremenna1 = inicializacnaMetoda1();
    private static int statickaPremenna2 = inicializacnaMetoda2();

    private static int inicializacnaMetoda1() {
        return 100;
    }

    private static int inicializacnaMetoda2() {
        return 200;
    }
}
```