Prednáška 1: Vhľad do objektovo-orientovaného programovania a programovacieho jazyka Java

Objektovo-orientované programovanie 2012/13

Valentino Vranić

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

20. február 2013

Obsah prednášky

1. Objektovo-orientované programovanie
2. Programovací jazyk Java
3. Operátory a riadenie vykonávania programu

Objektovo-orientované programovanie

Objekt

- OOP
- Programovanie pomocou objektov
- V zmysle OOP objekt je entita, ktorá má:[1]
  - stav – zahŕňa všetky vlastnosti objektu a ich hodnoty
  - správanie – ako objekt koná a reaguje v zmysle zmeny stavu a operácií, ktoré poskytuje
  - identitu – jednoznačná identifikácia objektu

[1] G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 1994.

Objekty a triedy

- Program sa uskutočňuje ako *interakcia objektov*
- Správanie objektu v tzv. v OO jazykoch so statickými typmi (ako je Java) definuje jeho *typ*
- Typ objektu sa označuje ako *trieda* (class)
- Niekedy sa zdá, že ide skôr o programovanie pomocou tried
- Iný prístup je v tzv. dynamických OO jazykoch

Stav a správanie objektu (1)

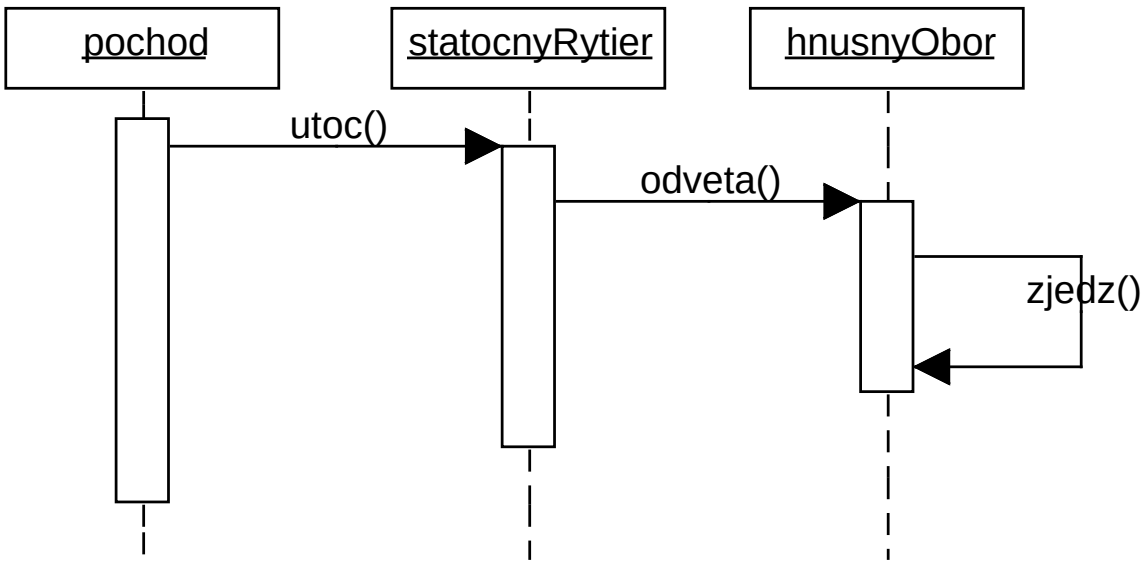
- Priblíženie OOP pomocou príkladu
- Predpokladajme, že vytvárame hru
- Jeden z objektov v hre je `statocnyRytier`
  - Stav:
    - `statocnyRytier.poloha`
    - `statocnyRytier.energia` – **int**
  - Správanie: `statocnyRytier.utoc()`

Stav a správanie objektu (2)

- Ďalší z objektov v hre je `hnusnyObor`
  - Stav:
    - `hnusnyObor.poloha`
    - `hnusnyObor.energia` – **int**
    - `hnusnyObor.hladny` – **boolean**
  - Správanie: napr. `hnusnyObor.odveta();`

Interakcia objektov

- Stret rytiera a obra ako interakcia objektov



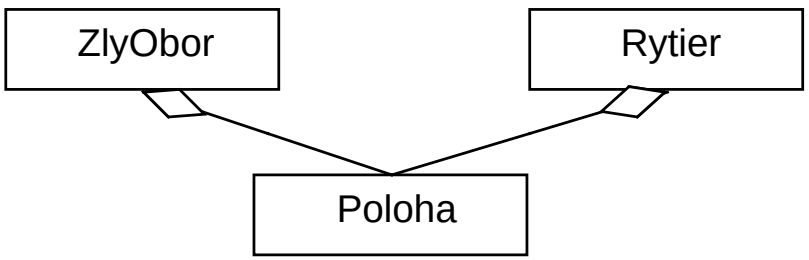
- Interakciu iniciuje objekt `pochod` (o ktorého detaily sa teraz nezaujímame)
- Objekty posielajú *správy* – volajú operácie, t.j. *metódy*

Typ objektu

- `hnusnyObor` je jeden zo zlých obrov – jeho typ je určený triedou `ZlyObor`
- `statocnyRytier` je objektom triedy `Rytier`
- `poloha` je objektom triedy `Poloha`

Agregácia

- Objekty tried `ZlyObor` a `Rytier` obsahujú objekty triedy `Poloha`



- Toto sa označuje ako *agregácia*
- Agregácia predstavuje spôsob tvorby *hierarchie*

Implementácia triedy – zapuzdrenie (1)

- Náčrt implementácie triedy `ZlyObor`:

```
class ZlyObor {
    Poloha poloha;
    int energia;
    boolean hladny;

    void odveta(Rytier r) {
        if (hladny)
            zjedz(r);
    }
    void zjedz(Rytier r) {
        int e = r.zistiEnergiu();
        r.znizEnergiu(e);
        zvysEnergiu(e);
    }
    int zistiEnergiu() { return energia; }
    void zvysEnergiu(int i) { energia = energia + i; }
    void znizEnergiu(int i) { energia = energia - i; }
}
```

Implementácia triedy – zapuzdrenie (2)

- *Zapuzdrenie* (encapsulation) – skrývanie informácií:
  - implementácia objektu má zostať skrytá
  - prístup prostredníctvom *rozhrania* (interface), ktoré tvoria vybrané metódy (čo je v tomto príklade mierne porušené)

Dedenie (1)

- `ZlyObor` je len jeden druh obrov
- *Zovšeobecnením* (generalizáciou) obrov je trieda `Obor`

```
class Obor {
    Poloha poloha;
    boolean hladny;
    int energia;

    void odveta(Rytier r) { r.znizEnergiu(1); }
    int zistiEnergiu() { return energia; }
    void zvysEnergiu(int i) { energia = energia + i; }
    void znizEnergiu(int i) { energia = energia - i; }
}
```

Dedenie (2)

- `ZlyObor` predstavuje *špeciálizáciu* triedy `Obor`

```
class ZlyObor extends Obor {
    void odveta(Rytier r) {
        if (hladny)
            zjedz(r);
    }
    void zjedz(Rytier r) {
        int e = r.zistiEnergiu();
        r.znizEnergiu(e);
        zvysEnergiu(e);
    }
}
```

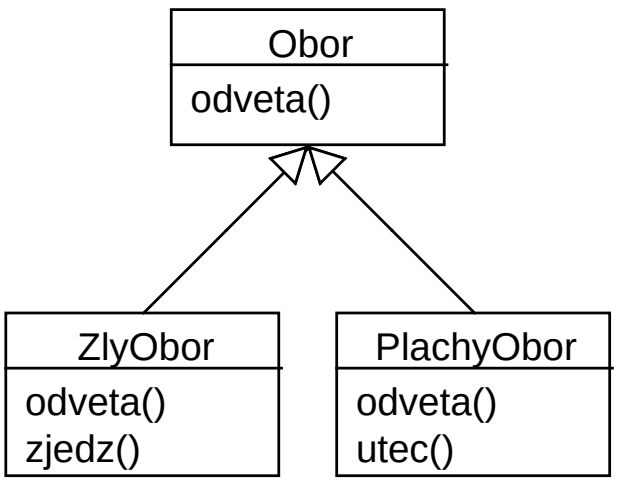
- `ZlyObor` *dedí správanie a štruktúru* triedy `Obor`
- `ZlyObor` *rozširuje a konkretizuje* triedu `Obor` – pridáva nové detaily do *abstrakcie* obra

Dedenie (3)

- `PlachyObor` je naozaj plachý

```
class PlachyObor extends Obor {
    void utec() { . . . }
    void odveta(Rytier r) {
        utec();
    }
}
```

- Dedenie predstavuje ďalší spôsob tvorenia hierarchie



Polymorfizmus (1)

- Aj objekty typu `PlachyObor`, aj objekty typu `ZlyObor` sú typu `Obor`

```
Obor o1;
Obor o2;

o1 = new ZlyObor();
o2 = new PlachyObor();

o1.odveta(r); // zje rytiera
o2.odveta(r); // utecie
```

Polymorfizmus (2)

- Pre hordu sto obrov pri ich strete (one by one) s rovnako početnou rytierskou výpravou môžeme napísať jednoducho

```
for (int i = 0; i < 100; i++) {
    obor[i].odveta(rytier[i]);
}
```

- Pole obor[] obsahuje premenné typu `Obor` – objekty, na ktoré ukazujú, sú odvodených typov

```
...
obor[47] = new ZlyObor();
obor[48] = new PlachyObor();
obor[49] = new Obor();
...
```

- Každý objekt v poli sa predsa bude správať podľa svojho skutočného typu
- Nepotrebujeme žiadne podmienené príkazy
- To je *polymorfizmus*: každý objekt sa môže uplatniť na mieste objektu hociktorého z jeho nadtypov

Objektovo-orientované programovanie a jazyk

- Pozrieme sa na OO z pohľadu Javy, neskôr sa vrátime k princípom
- Jazyk – najdôležitejší prostriedok vyjadrovania ľudských myšlienok
- Softvér sa v konečnom dôsledku vyjadruje v programovacom jazyku

Programovací jazyk Java

Čo je Java

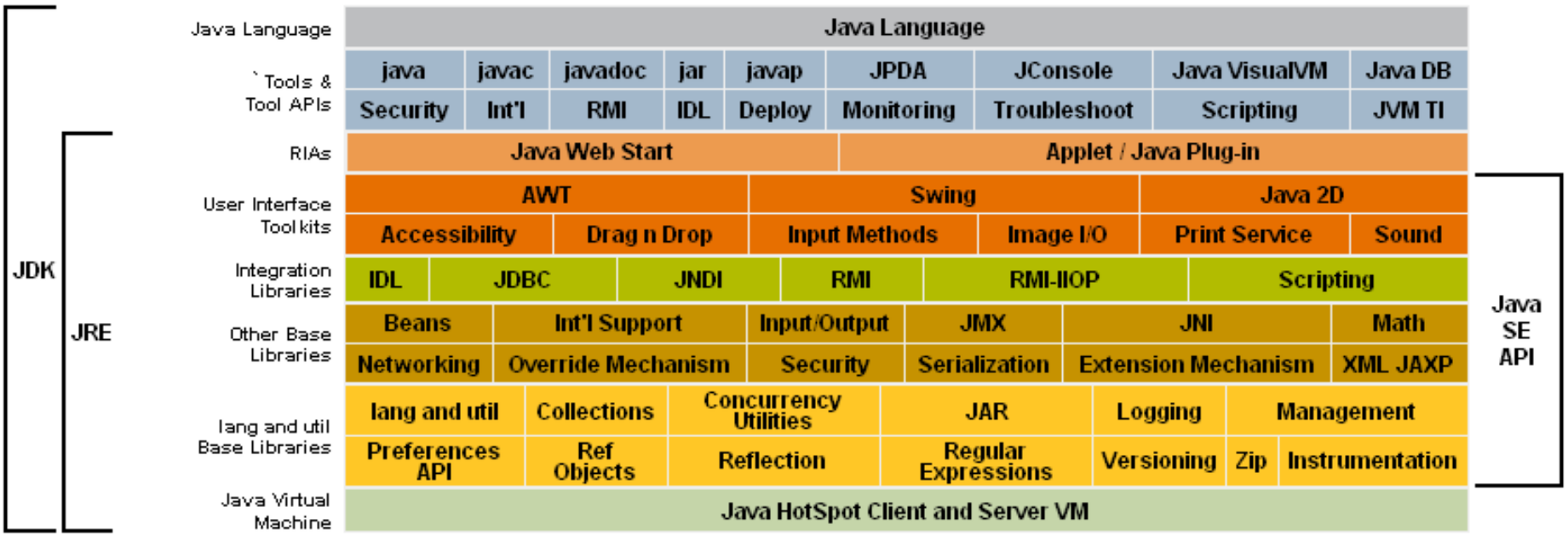
- Java – objektovo-orientovaný programovací jazyk
- Programy v Jave sa kompilujú (prekladajú)
  - výsledok je tzv. *bytecode* – bajtkód
  - raz napísaný programy v Jave sa dá vykonávať všade
  - t.j. malo by platiť „Write once, run anywhere“ (hoci niekedy je to skôr „Write once, debug everywhere“)
- Java je teda nezávislá od platformy, lebo...

Java *je* platforma

- Platforma – Java runtime environment (JRE):[2]
  - Java application programming interface (Java API)
  - Java virtual machine (JVM)
  - ďalšie komponenty potrebné na vykonávanie Javy
- Dostupné sú rôzne platformy podľa prostredia:
  - Java Micro Edition (Java ME) – pre prostredia s obmedzenými zdrojmi
  - Java Standard Edition (Java SE) – pre pracovné stanice
  - Java Enterprise Edition (Java EE) – pre rozsiahle distribuované prostredia
- Java Software Development Kit (JDK)
  - JRE + nástroje na vývoj

[2] `http://www.oracle.com/technetwork/java/javase/tech/`

Java SE 7



http://download.oracle.com/javase/7/docs/

Vývoj Javy

- Vývoj začal v Sun Microsystems v roku 1990[3] – od roku 2010 Oracle
- Meno Java vzniklo v roku 1994 (pôvod nie je celkom známy; výslovnosť *java*/*džava*)
- Významné verzie:
  - 1.0 (1996) – uvádzacia verzia
  - 1.1 (1997) – rozsiahle zmeny (zavedenie vnútorných tried...)
  - 1.2 (1998) – vznik tzv. Java 2
  - 1.3 (2000) – menšie zmeny a opravy
  - 1.4 (2002) – nový vstupno/výstupný systém (NIO)
  - J2SE 5.0 (Tiger, 2004) – významné rozšírenie jazyka
  - Java SE 6 (Mustang, 2006) – zmeny v API a vývojových nástrojoch, ale bez rozšírení jazyka
  - Java SE 7 (Dolphin, 2011) – rad menších zmien v jazyku[4]
  - Java SE 8 – vo vývoji[5]

[3] http://en.wikipedia.org/wiki/Java_programming_language
[4] http://www.eclipse.org/jdt/ui/r3_8/Java7news/whats-new-java-7.html
[5] https://blogs.oracle.com/javaone/resource/java_keynote/slide_16_full_size.gif

Integrované vývojové prostredia pre Javu

- Niektoré prostredia pre Javu:
  - Eclipse, `http://eclipse.org/`
  - NetBeans, `http://netbeans.org/`
  - JBuilder, `http://www.embarcadero.com/products/jbuilder`
  - JCreator, `http://www.jcreator.com/`

Programovací jazyk Java

- Syntax založená na jazykoch C a C++
- K objektom sa pristupuje vždy cez referenciu

```
String s;
```

- Referencia pred použitím musí byť inicializovaná

```
String s = new String("asdf");
```

- Skráteny tvar špeciálne pre String:[6]

```
String s = "asdf";
```

[6] Prináša pamäťovú optimalizáciu – podrobnejšie na nasledujúcej prednáške

Primitívne typy

- **char**, **byte**, **short**, **int**, **long**, **float**, **double**, **boolean**, **void**
- Pre efektívnejšie narábanie s pamäťou: vytvárajú sa na zásobníku (stack), a nie na hromade (heap)
- Podobne ako v C, ale veľkosť pre každý nezávisí od prostredia
- Pre každý primitívny typ je definovaná obaľovacia trieda
  - **char** – `Character`
  - **int** – `Integer`
  - pre ostatné typy názov zostáva rovnaký, len začína veľkým písmenom

Uvoľňovanie pamäte

- Rozsah platnosti (scope) – vymedzený skupinovými zátvorkami

```
{
    String s = new String("a string");
} // odtiaľto s už nie je
```

- V Jave sa objekty nerušia explicitne
- Objekty, na ktoré sa nikto neodvoláva zruší zberač smetí (garbage collector)

Trieda

- Určuje typ objektu, a tým aj jeho správanie
- Trieda má názov a telo

  class `NazovTriedy` { /* *telo triedy* */ }

- Trieda môže obsahovať:
  - atribúty (v Jave *polia* – *fields*; niekedy aj *premenné*) – údaje
  - metódy – operácie

Atribúty

- Trieda obsahujúca len údaje – podobná **struct** v C

```
class Student {
    String meno;
    boolean zapisany;
}
```

- Atribúty môžu byť primitívneho typu, ale aj referencie na objekty
- Objekt vytvoríme pomocou operátora **new**

```
Student s = new Student();
```

Prístup k atribútom

- K atribútom sa pristupuje pomocou operátora .

```
s.zapisany = false;
```

- Prístup funguje aj pre objekty v rámci daného objektu

```
class Zaradenie {
    String fakulta;
    String odbor;
}
class Student {
    String meno;
    boolean zapisany;
    Zaradenie zaradenie;
}
```

- Cez študenta sa dostaneme k jeho zaradeniu:

```
s.zaradenie.fakulta = "FIIT";
s.zaradenie.odbor = "Informatika";
```

- Inicializácia atribútov je zabezpečená, ale lokálnych premenných (v metódach) *nie*

Metódy

- Metódy sú podobné funkciám v C

```
Typ nazovMetody(/* parametre */) {
    /* telo */
}
```

- Kľúčové slovo **return** slúži na:
  - vrátenie hodnoty

```
int storage(String s) {
    return s.length() * 2;
}
```

  - ukončenie metódy

```
void nothing() {
    return;
}
```

Balíky

- Balíky (packages) sú (trochu) podobné pridávaniu súborov v C (`include`)
- Použitie sa deklaruje kľúčovým slovom **import**
  - deklarácia použitia len jednej triedy:

    import `java.util.ArrayList;`

  - deklarácia použitia celého balíka:

    import `java.util.*;`

- Okrem **java.lang** ostatné štandardné balíky Javy treba explicitne importovať
- Štruktúra a dokumentácia všetkých štandardných balíkov sa dá prehliadať na `http://docs.oracle.com/javase/7/docs/api/`

Statické atribúty

- Statické atribúty jestvujú ako súčasť triedy
- Statický atribút zdieľajú všetky objekty danej triedy

```
class StaticTest {
   static int i = 47;
}
```

- Skúška:

```
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
StaticTest.i++; // st1.i = st2.i = 48
```

- Po inkrementácii aj `st1.i`, aj `st2.i` budú mať hodnotu 48

Statické metódy

- Statické metódy sa dajú volať priamo, bez vytvárania objektu

```
class StaticFun {
   static void incr() { StaticTest.i++; }
}
```

- Volanie cez triedu:

```
StaticFun.incr();
```

- Statické metódy sa dajú volať aj prostredníctvom objektov – účinok je rovnaký

```
StaticFun sf = new StaticFun();
sf.incr();
```

Hello, world!

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

- Program treba uložiť do súboru s názvom triedy: `HelloWorld.java`
- Spustenie prekladu: `javac HelloWorld.java`
- Spustenie programu: `java HelloWorld`

Komentár a vnorená dokumentácia

- Komentár ako v C++

```
// Jednoriadkovy komentar

/* Bezny komentar
   (ako v jazyku C)
*/
```

- Vnorená dokumentácia
  - špeciálne vyznačené časti v rámci komentára
  - Javadoc vygeneruje dokumentáciu v HTML
  - *preštudujte v TiJ*

Operátory v Jave

- Väčšinou vám budú známe z jazyka C
  - *preštudujte v TiJ*
- Operátory fungujú väčšinou len pre primitívne typy
  - =, == a != možno použiť pre hocijaké objekty
  - pre triedu String možno použiť ešte aj + a +=
- Priorita operátorov je podobná ako v C
  - časom si zvyknete (netreba vedieť spamäti)
  - pri pochybnostiach použite zátvorky

Priradenie

- Premenné primitívneho typy
  - priraďujú sa hodnoty (ako v C)
- Objekty:
  - priraďujú sa referencie – nie hodnoty
  - referencie potom ukazujú na ten istý objekt
  - zmena cez hociktorú z týchto referencii vplýva na ten istý objekt

Priradenie a objekty

- Nech je daná trieda:

```
class Number {
    int i;
}
```

- Vyskúšajme priradenie s objektmi:

```
Number n1 = new Number();
Number n2 = new Number();

n1.i = 1;
n2.i = 2;
System.out.println(n1.i + " " + n2.i); // 1 2

n1 = n2;
System.out.println(n1.i + " " + n2.i); // 2 2

n1.i = 3;
System.out.println(n1.i + " " + n2.i); // 3 3
```

Objekty ako parametre

- Objekty sa môžu vyskytovať ako parametre

```
static void f(Number n) {
    n.i = 0;
}
```

- Prenášajú sa referencie, nie objekty

```
Number x = new Number();

x.i = 1;
System.out.println(x.i); // 1

f(x);
System.out.println(x.i); // 0
```

Operátory a riadenie vykonávania programu

Matematické operátory

- Matematické operátory sú ako v C
- Základné operácie: $+$, $-$, $*$, / a %
- Skrátený zápis: *op*= (napr. $+=$)
- Unárne operátory: $+$ a $-$
- Inkrementácia a dekrementácia: $++$ a $--$

Relačné operátory

- Zápis ako v C: ==, !=, >, <, >= a <=
- Výsledok porovnávania je však typu boolean
- Pri objektoch sa porovnávajú referencie – nie obsah
  - na porovnávanie obsahu objektov jestvuje metóda `equals()`
  - k nej sa dostaneme na ďalších prednáškach

Logické operátory

- Znovu podobne ako v C: &, | a !
  - Použiteľné len pre boolean hodnoty
  - Výsledok je boolean
- Operátory && a ||
  - Podmienené verzie operátorov &, |
  - Výraz sa vyhodnocuje len kým nie je jasná jeho hodnota
  - Príklad:

```
static boolean test(int val) {
    System.out.println("Porovnavam" + val);
    return val < 2;
}
```

  - Vyhodnotí sa len prvá časť podmienky:

```
if(test(3) && test(1))
    ; // Porovnavam 3
```

Porovnávacie operátory na bitoch

- Znovu ako v C: &, |, ^ a ~
- Skrátený zápis: *op*= (okrem unárnho operátora ~)
- Použiteľné pre celočíselné hodnoty

Posúvacie operátory na bitoch

- << a >> ako v C
- K tomu ešte >>> (pri posúvaní vkladá nuly bez ohľadu na znamienko)
- Použiteľné pre celočíselné hodnoty
- Použitie na **char**, **byte**, **short** a **int** dáva **int** výsledok; **long** dáva **long** výsledok
- Skrátený zápis: *op*=
- Pozor pri aplikácii >>> na **byte** a **short**

Ternárny if-else operátor

- Známy z jazyka C
- Výraz:

```
return i < 10 ? i * 100 : i * 10;
```

je ekvivalentný s:

```
if (i < 10)
    return i * 100;
else
    return i * 10;
```

Ďalšie operátory

- Operátor ,
  - v Jave sa používa len vo **for** slučkách
- Operátor spájania reťazcov znakov +
- Casting (zmena typu)

```
void casts() {
    int i = 200;
    long l = (long)i;
    long l2 = (long)200;
}
```

Literály

- Hodnoty zadávané priamo v programe
- Prekladač väčšinou dokáže vybrať typ
- Niekedy typ treba vyjadriť explicitne:

```
char c = 0xffff; // max. hodnota pre char hexadecimalne
int i3 = 0177; // oktálna hodnota (začína nulou)
float f4 = 1e-45f; // exponenciálny zápis
```

Riadenie vykonávania programu

- Ako v C (*preštudujte v OJA a TiJ*)
  - **if-else**
  - **return**
  - **while**
  - **do-while**
  - **for**
  - **break** a **continue**
  - **switch-case**
- Java má tiež návestia, ale nemá **goto** (dá sa však simulovať – vysvetlené v TiJ)

Sumarizácia

Sumarizácia

- Vhľad do objektovo-orientovaného programovania prostreníctvom príkladu
- Mechanizmy objektovo-orientovaného programovania: agregácia, zapuzdrenie, dedenie a polymorfizmus
- Java ako jazyk a platforma
- Niektoré prvky jazyka Java a súvis s jazykom C

Čítanie

- Dnešná prednáška:
  - OJA, kapitola 2, kapitola 3: časti 3.1–3.7 a 3.11
  - TiJ, kapitola 3: operátory a riadenie vykonávania programu – mali by ste poznať z jazyka C; záležitosti ohľadom testovacieho systému vynechajte
  - TiJ, kapitola 2: vnorená dokumentácia nebola prednášaná – pozrite pre praktické použitie (úloha na cvičeniach)
- Ďalšia prednáška bude o elementárnom programovaní v Jave – OJA, kapitoly 3 a 4