Programovací jazyk Java

Vlastnosti a použití jazyka Java Implementace vybraných objektových principů

přednáška č. 1

Cíle předmětu

- Naučit se prakticky aplikovat teoretické poznatky z předmětů *Algoritmizace* a *Základy objektového návrhu* prostřednictvím konkrétního objektového programovacího jazyka.
- Seznámit se s oblíbeným moderním programovacím jazykem, který svou šíří použití pokrývá oblast od chytrých karet přes aplikace pro mobilní zařízení, pro osobní počítače až po robustní, webově orientované informační systémy.
- V předmětu PJJ je pozornost zaměřena na správnou aplikaci principů objektového programování.
  V navazujícím volitelném předmětu *Java aplikace* se studenti naučí vytvářet webové aplikace běžící na straně serveru a aplikace pro mobilní telefony.

Podmínky ukončení předmětu

- Co bude hodnoceno:
  - Dva průběžné testy ve cvičeních (lze získat 2× 10 bodů).
  - Zkouškový test rozdělený na teoretickou a praktickou část (lze získat 80 bodů).
- Podmínky zkoušky:
  - Min. 60 bodů celkově.
- Hodnocení zkoušky – podle dosažených bodů:
  - od 90 bodů – A od 83 bodů – B od 75 bodů – C
  - od 68 bodů – D od 60 bodů – E < 60 bodů – F

Vyučující

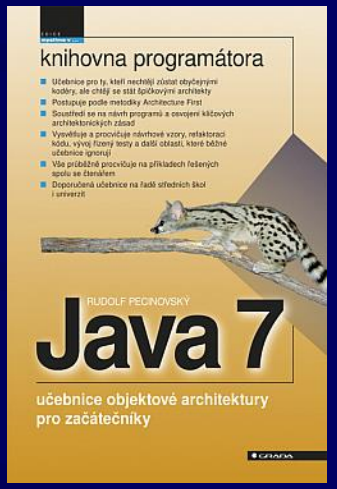
- Přednášky, cvičení
  - Ing. Petr Jedlička, Ph.D.
    petr.jedlicka@mendelu.cz

Studijní zdroje

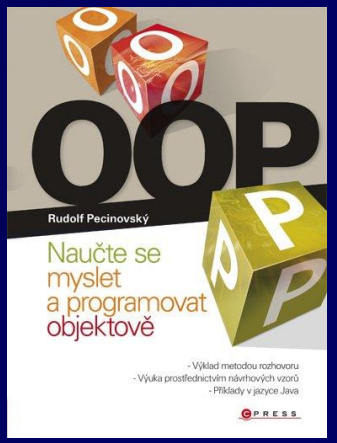
- JDK 21 Documentation
  - Java 21 API
- Základy programování – videokurz (40 lekcí)
- Objektově orientované programování v Javě
- Java – Česká online učebnice (vybrané lekce zdarma)
- Seriál *Java pro začátečníky*
- The Java Tutorials

Studijní zdroje

- Pecinovský, R.: Java 7 – Učebnice objektové architektury pro začátečníky. Praha: Grada Publishing, 2012. 496 s. ISBN 978-80-247-3665-5.



- Pecinovský, R.: OOP – Naučte se myslet a programovat objektově. Brno: Computer Press, 2010. 576 s. ISBN 80-251-2126-9.



Studijní zdroje

- Pecinovský, R.: Návrhové vzory. 1. vyd. Brno: Computer Press, 2007. 527 s. ISBN 978-80-251-1582-4. Recenze.

- Beck, K.: Programování řízené testy. 1. vyd. Praha: Grada, 2004. 204 s. ISBN 80-247-0901-5.

Obsah 1. přednášky

- Vlastnosti a použití jazyka Java
- Implementace zásad objektově orientovaného programování v jazyce Java

Co je Java?

- Java je moderní, univerzální, objektově orientovaný programovací jazyk, vyvinutý firmou Sun Microsystems určený zejména pro vývoj internetových aplikací. Spolu s C++ a Pythonem je Java jedním ze tří nejpoužívanějších jazyků na světě.
- Název jazyka odvozen od slangového označení kávy (viz logo).
- Původní tvůrce Javy – firma Sun – Javu charakterizoval jako *jednoduchý, objektově orientovaný, distribuovaný, interpretovaný, robustní, bezpečný, nezávislý na architektuře, přenositelný, výkonný a víceprocesní jazyk.*

Proč se učit zrovna Javu?

1. Je syntakticky jednoduchá.
2. Sama se stará o uvolňování nepotřebné paměti.
3. Javovské programy jsou přenositelné (tj. stejný program lze bez úprav pustit na libovolném operačním systému).
4. Je použitelná pro projekty každého rozsahu.
5. Je vhodná pro začátečníky – chyby v programech se dají (obvykle) snadno vyhledat.
6. Je použitelná na všech typech zařízení od vestavných po velké servery.
7. Je výkonná, zejména v porovnání s interpretovanými jazyky (PHP, Ruby).

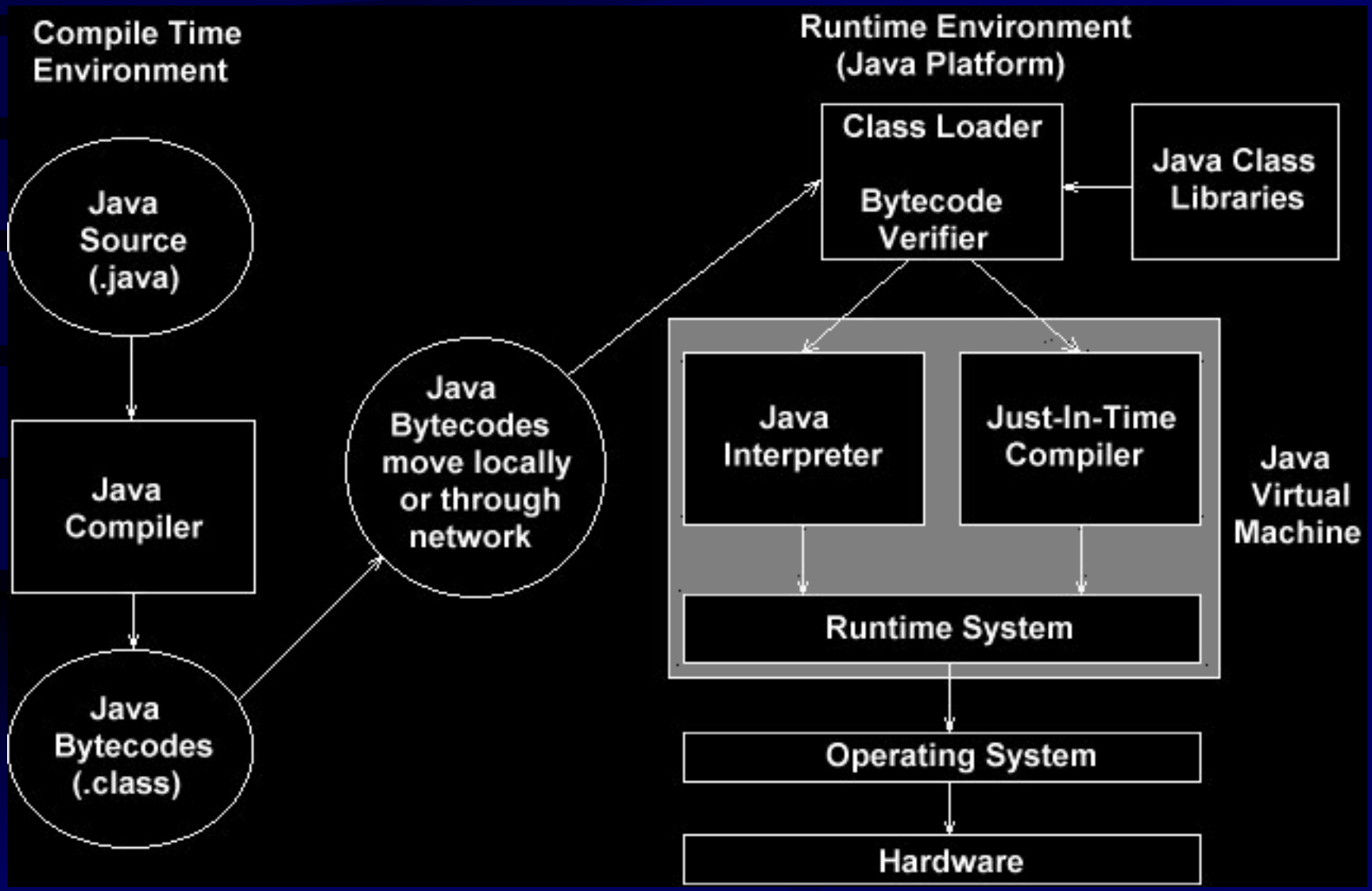
Proč se učit zrovna Javu?

8. Neumožňuje udělat některé programátorské zločiny proti lidskosti.
9. Je široce používaná.
10. V případě nesnází není problém nalézt někoho, kdo Vám poradí.
11. Její znalost je výhodou na trhu práce.
12. Vždy se můžete inspirovat kódem jednoho z tisíců otevřených projektů v Javě.
13. Pro vývoj v Javě existují velmi kvalitní vývojová prostředí a nástroje usnadňující práci.

K čemu se Java nehodí?

- Real-time systémy – systémy, kde je zapotřebí garantovat odezvu v daném (vesměs velmi malém) časovém horizontu. V Javě nelze garantovat kvůli automatické správě paměti.
- Drivery, součásti operačních systémů, výpočetně velmi náročné aplikace. Zde bývá závislost na operačním systému nebo přímý přístup k hardwaru. Ač je JIT (Just-In-Time) kompilovaná, poskytuje nižší výkon než jazyk C (který je pro tyto účely výrazně vhodnějším kandidátem).

Kompilace a běh programu



Java Runtime Environment – JRE (Java platforma)

Prostředí pro běh programů v Javě, má 2 části:

1. **Java Virtual Machine – JVM**
   Abstraktní počítač – virtuální stroj, tvořen runtime systémem (realizuje vazbu na hardware) a interpretem (vykonává bytový kód). Pro urychlení může být interpret volitelně nahrazen JIT (Just-In-Time) kompilátorem, který při běhu programu provádí nejprve překlad do strojového kódu příslušného procesoru.
2. **Java Core API**
   Aplikační programové rozhraní. Základní knihovny pro psaní programů. Výhoda: tyto knihovny nemusí být s programem distribuovány, neboť jsou povinnou součástí Java Platformy.

Java Development Kit – JDK

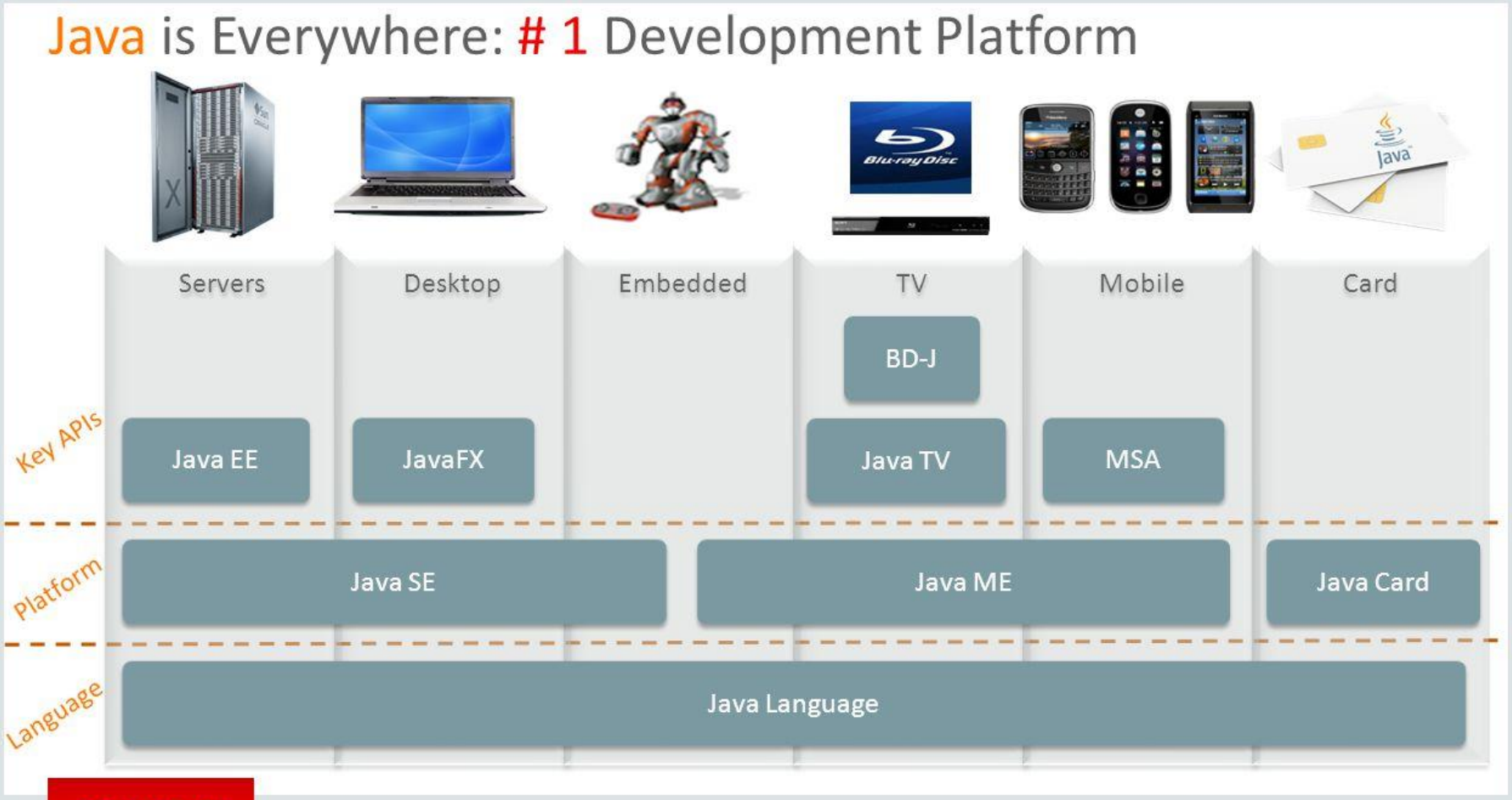
- Obsahuje JRE + překladač + další vývojové nástroje.
- Číslování verzí JRE a JDK je shodné.
- Původně byly verze Javy číslovány od 1.0 po 1.4. Verze 1.5 (v r. 2004) už se od počátku označovala jako 5, následovaly verze 6, 7 atd.
- Některé verze Javy jsou značeny LTS (Long Time Support) – jedná se o verzi, pro kterou jsou dlouhodobě vydávány opravy a updaty.

Java platformy

- Java SE (Standard Edition) je platforma pro psaní klasických programů pro osobní počítače (desktopových aplikací) → náplní tohoto předmětu
- Java EE (Enterprise Edition) je platforma pro psaní rozsáhlejších (distribuovaných) aplikací. V podstatě se jedná o rozšíření platformy Java SE.
- Java ME (Micro Edition) je určena speciálně k psaní aplikací pro mobilní zařízení (telefony, tablety apod.).
- Java Card – pro aplikace provozované v rámci tzv. „chytrých“ karet (např. platební a kreditní karty atp.)
- Java TV – platforma pro tvorbu aplikací pro televizory a set top boxy

Rozdíl mezi Javou pro Android a Javou

Přehled použití Java platforem



Tvorba aplikace

- Překlad se provádí kompilátorem **javac**, zadává se celý název souboru. Spuštění se provádí příkazem **java**, zadává se název souboru s bytekódem bez přípony. Příklad:
- **Zdrojový text:** `Program.java`
- **Překlad:** `javac Program.java`
- **Výsledek:** `Program.class`
- **Spuštění:** `java Program`

Způsob zápisu identifikátorů

- **Třídy a rozhraní**
  - identifikátor začíná velkým písmenem, ostatní jsou malá. Pokud je utvořen z více slov, je počáteční písmeno každého slova velké. Např. `String`, `StringBuffer`.
- **Metody a proměnné**
  - pouze malá písmena, pokud z více slov, pak začátek dalšího velkým písmenem, např. `pocet`, `pocetPrvku`, `getSize()`
- **Balíky**
  - pouze malá písmena, ve složených názvech slova oddělena tečkou. Např. `java.lang`, `java.awt.image`
- **Konstanty**
  - pouze velká písmena, pokud z více slov → podtržítka. Např. `PI`, `MAX_VALUE`

Objekt

- Jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných **objekty**.
- Objekty si pamatují svůj **stav** a navenek poskytují **operace** (přístupné jako metody pro volání). **Co je to stav objektu?**
- **Stav objektu** = hodnoty atributů.
- Objekty mezi sebou komunikují pomocí **zpráv**. **Z čeho se skládá zpráva?**
- **Zpráva** = identifikátor objektu + identifikátor metody + případné parametry metody.
- **Otázka: Může objekt poslat zprávu sám sobě?**

Třída

- Abstrakce objektu, která v programu podchycuje na obecné úrovni podstatu všech objektů podobného typu.
- Objekt je instancí třídy. Co to znamená?
- Objekt je konkrétní realizací obecného vzoru definovaného v příslušné třídě.
- Všechny instance jedné třídy mají stejnou strukturu (atributy a metody). V čem se tedy liší?
- Instance téže třídy se mohou lišit ve svých stavech.

Třída v Javě

- ```
  class MojeTrida {
      tělo třídy
  }
  ```
- Zdrojový kód třídy zpravidla ukládáme do souboru se stejným názvem (včetně velikosti písmen) a příponou *java*. Např. `MojeTrida.java`. Tato shoda je nutná u veřejných tříd (hlavička třídy začíná slovem `public`), které jsou pak dostupné i mimo svůj balíček.
- V jednom souboru může být zapsáno i **více tříd za sebou** (každá je pak zkompilována do samostatného *.class* souboru).
- **Třídy lze i vnořovat**, pro vnitřní třídy však platí určitá **omezení**.
- Lze použít i **anonymní třídy** (beze jména), definované až v místě použití.
- Třída obsahuje především definice metod a deklarace proměnných.
  ```

Základní (primitivní) datové typy

- **celočíselné**
  - pouze znaménkové
  - byte (1 B), short (2 B), int (4 B), long (8 B)
- **reálné**
  - float (4 B), double (8 B)
- **znakové**
  - char (2 B) – Java implicitně používá kódování Unicode
- **logické**
  - boolean
- **prázdný typ** `void`

Deklarace proměnných

- Formát: `datovýTyp jménoProměnné;`
- Příklad: `int i;`
- Po **deklaraci** by ještě před použitím proměnné mělo dojít k její **inicializaci**:
  - při deklaraci: `int i = 5;` nebo
  - později: `i = 5;`
- Překladač nepovolí použití neinicializované proměnné jinde než na levé straně přiřazovacího příkazu

Zápis čísel

- Zapsané celé číslo je automaticky chápáno jako číslo typu `int`, reálné číslo jako číslo typu `double`.
- Chceme-li, aby bylo zapsané celé číslo chápáno jako typ `long`, zapíšeme za něj „L“:
  - `int x = 5;` je OK, ale `int x = 5L;` nelze.
- Chceme-li, aby bylo zapsané reálné číslo chápáno jako typ `float`, zapíšeme za něj „F“ (implicitně je double):
  - `float x = 5.5;` je chyba, `float x = 5.5F;` je OK.
- Můžeme používat i čísla v šestnáctkové soustavě, před číslo umístíme „0x“ nebo „0X“ :
  - `int j = 0x1c;` // číslo 28 v šestnáctkové soustavě.
- Můžeme používat i čísla v osmičkové soustavě, před číslo umístíme nulu:
  - `int j = 034;` // číslo 28 v osmičkové soustavě.

Deklarace konstant

- Jako deklarace proměnné, navíc se použije klíčové slovo `final`
- Příklad: `final int MAX = 10;`
- Konstantě poté již nelze přiřadit žádnou hodnotu (ani stejnou), s výjimkou:
-
```
final int MAX;
...
MAX = 10;
```

Výraz, přiřazení, příkaz

- **Terminologie:**
  - **L-hodnota = něco, co má adresu v paměti (písmeno L znamená, že se nachází na levé straně přiřazovacího příkazu).**
    Co JE L-hodnotou: proměnná `x`.
    Co NENÍ L-hodnotou : literál `526`, výraz `(x + 9)`

česky	anglicky	symbolicky	prakticky
výraz	expression	výraz	x * 5 + 8
přiřazení	assigment	L-hodnota = výraz	y = x * 5 + 8
příkaz	statement	L-hodnota = výraz;	y = x * 5 + 8;

Operátor přetypování (konverze)

- Jako operátor použijeme název typu v závorce (tzv. explicitní konverze)
- Příklad:

```
double d = 5;
int i;
i = (int) d;
```

- V případě rozšiřující konverze možno použít implicitní konverze (`d = i;`)
- Rozšiřující konvence pro základní datové typy: byte $\rightarrow$ short $\rightarrow$ int $\rightarrow$ long $\rightarrow$ float $\rightarrow$ double
- Přetypování má nejvyšší prioritu

Aritmetické operátory

- **Unární**
  - `+`, `-` (prefix, tedy např. `+5`, `-j`)
  - `++`, `--` (prefix nebo postfix, tedy např. `x++`, `--i`)

    Nelze však napsat např. 5++, protože zde musí být l-hodnota.
- **Binární**
  - `+`, `-`, `*`, `/`, `%` (dělení modulo)
- **Přiřazovací**
  - `L-hodnota = l-hodnota operátor výraz`
  - lze zkrátit na `L-hodnota operátor= výraz`
  - např: `x = x / z` lze zapsat jako `x /= z`

Další operátory

- **Relační**
  - `<, >, <=, >=, ==, !=`
- **Logické**
  - `&&` – logické A
  - `||` – logické NEBO
  - `!` – negace
- **Bitové**
  - `^` – bitová negace (např. 9 ^ 3 = 10, 9 ^ 7 = 14)
  - `&` – bitové A (např. 9 & 3 = 1, 13 & 11 = 9)
  - `|` – bitové NEBO (např. 9 | 3 = 11, 9 | 7 = 15)

Priorita operátorů

Category	Operators
Primary	`x.y f(x) a[x] x++ x-- new typeof checked unchecked`
Unary	`+ - ! ~ ++x --x (T)x`
Multiplicative	`* / %`
Additive	`+ -`
Shift	`<< >>`
Relational and type testing	`< > <= >= is as`
Equality	`== !=`
Logical	`AND &`
Logical	`XOR ^`
Logical	`OR |`
Conditional	`AND &&`
Conditional	`OR ||`
Conditional	`?:`
Assignment	`= *= /= %= += -= <<= >>= &= ^= |=`

Zápis třídy

- Třída je definována v souboru `(Xy.java)`.
- Jeden soubor může obsahovat lib. počet neveřejných tříd.
- Jeden soubor může obsahovat max. jednu veřejnou třídu.
- Jméno souboru odpovídá jménu veřejné třídy.
- Zápis třídy: `[public] class Jmeno {}`

Zápis třídy – příklady

```
// Soubor Point.java
public class Point {
    ...
}

// Soubor Y.java
class X {...}
class Z {...}
public class Y {...}
```

Organizace tříd do balíků

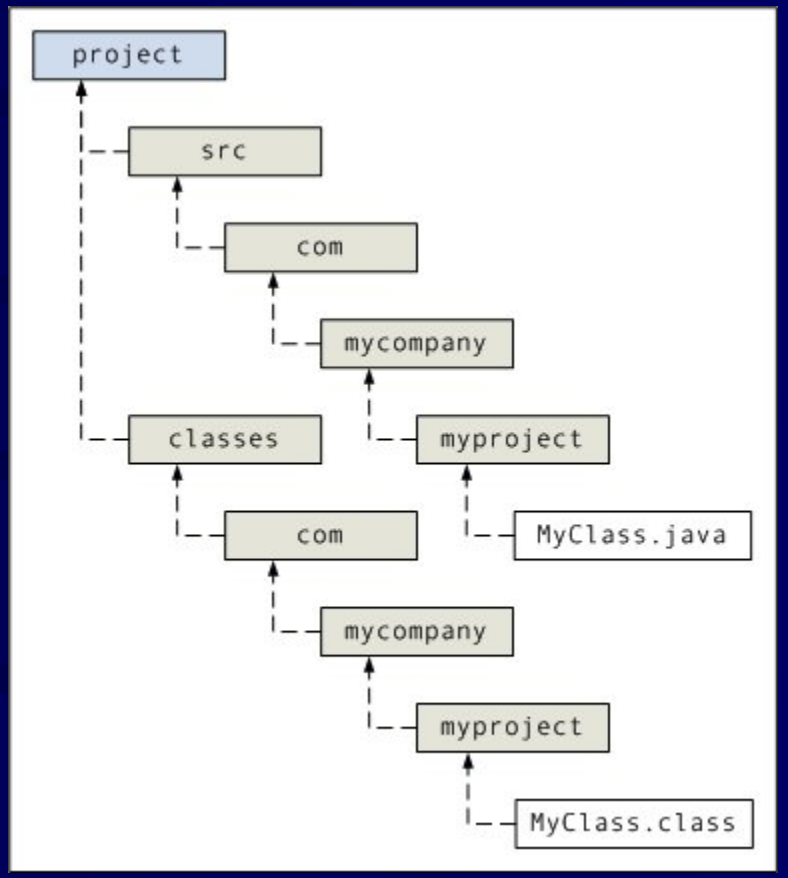
- Třídy a rozhraní organizujeme do balíků (packages)
- V balíku jsou vždy umístěny „související“ třídy.
- Jméno balíku + jméno třídy = plně kvalifikované jméno třídy a tedy jednoznačná identifikace třídy.
- Třídy ze stejného balíku jsou navzájem viditelné.
- Třídy z různých balíků je třeba:
  - adresovat plně kvalifikovaným jménem (tzn. včetně jména balíčku)

  NEBO

  - před hlavičkou třídy importovat.

Organizace tříd do balíků

- Balík složen z dílčích jmen, která jsou oddělena tečkami.
- Fyzická reprezentace balíku je adresářová struktura.
- Ve cvičeních tomto předmětu budeme pojmenovávat balíčky podle tohoto vzoru:
- `cz.mendelu.pef.pjj.xnovak.cv1`



Import třídy

- Třídy, rozhraní a další prvky z balíku `java.lang` lze používat bez importu.
- Pokud je třeba použít jinou třídu, než z aktuálního balíku nebo balíku `java.lang`, je třeba provést import (nebo uvést název třídy včetně názvu balíku).
- Klíč. slovo `import`:
  - `import nazev.baliku.Trida;` // import 1 třídy
  - `import nazev.baliku.*;` // import všech tříd balíku

Struktura souboru zdrojového kódu

1. Sekce balíku – nepovinná: `package balik2;`
2. Sekce importu – nepovinná:
   - `import balik.Trida;`
   - `import balik.*;`
3. Sekce zápisu definice tříd.

Příklad:

```
package cz.mendelu.pef.pjj.xnovak.cv2;

import cz.mendelu.pef.pjj.xnovak.cv1.*;
import java.io.File;

class Xyz {...}
```

Metody

- **Program v Javě obsahuje jednu nebo více definic metod.**
- **Metody v OOP zastupují podprogramy typu *procedura* a *funkce*. Jaký je rozdíl mezi procedurou a funkcí?**
- **V těle jedné metody nesmíme definovat jinou metodu, můžeme ji pouze volat.**

Definice metody

- Zahrnuje **hlavičku** (typ návratové hodnoty, jméno metody, příp. jména a typy formálních parametrů) **a tělo** metody.
- Minimální syntaxe:

```
NávratovýTyp jménoMetody ([p1[, p2[, p3[, …]]]])
{
    // tělo metody
}
```

- Metoda s návratovým typem (něco jako funkce): návratový typ je buď primitivní datový typ, třída, nebo rozhraní (budeme probírat později).
- Metoda s návratovým typem končí zavoláním příkazu `return hodnota;` (resp. `return výraz;`)

Metoda s návratovým typem

- Obdoba funkce z procedurálního programování.
- Návratový typ je buď primitivní datový typ, třída, nebo rozhraní (budeme probírat později).
- Metoda s návratovým typem končí povinným zavoláním příkazu `return hodnota;` (resp. `return výraz;`)
- Příklad:

```
int max(int a, int b) {
  if (a > b)
    return a;
  else
    return b;
}
```

Metoda bez návratového typu

- Obdoba procedury z procedurálního programování.
- Ve skutečnosti se nejedná o metodu bez návratového typu, ale bez návratové hodnoty. Jako návratový typ se uvádí typ `void` (angl. *prázdno, prázdnota*).
- Metoda končí provedením vších příkazů, případně kdykoliv zavoláním nepovinného příkazu `return;`.
- Příklad:

```
void vypis() {
  int a, b, c, i = 1, j = 2;
  a = max(i, j);
  b = max(i + 3, j);
  c = max(i, 5);
}
```

Metoda bez parametrů

- Musí být definována i volána včetně kulatých závorek (v tomto případě prázdných).
- Např. metoda

```
int secti2CislaZeVstupu() {
    int a, b;
    a = ctiInt();
    b = ctiInt();
    return (a + b);
}
```

- je volána

```
j = secti2CislaZeVstupu();
```

Metoda s více parametry různých typů

- Parametry se zapisují **jednotlivě** včetně typů, oddělují se čárkami. Pořadí parametrů může být v hlavičce libovolné, při volání však musí být dodrženo.
- **Správně** definované metody

```
double secti(int a, double b) {
    return a + b;
}

int secti(int a, int b) {
    return a + b;
}
```

- **Nesprávně** definovaná metoda

```
int secti(int a, b) {
    return a + b;
}
```

Rekurzivní metody

- Žádná zvláštní omezení, metoda prostě ve svém těle volá sama sebe (přímá rekurze) nebo se více metod volá cyklicky navzájem (nepřímá rekurze).
-
```
long fakt(long n) {
    if (n > 1)
        return n * fakt(n - 1);
    else
        return 1;
    }
}
```

Parametry metod

- V případě odlišnosti typu skutečného a formálního parametru lze počítat pouze s implicitní rozšiřující konverzí, zužující konverzi je třeba zadat explicitně.
- Pokud předáváme jako parametry primitivní datové typy, předávají se **výhradně hodnotou**.
- ```
  public class Konverze {
    int konv1(double d) {
      return (int) d;
    }
    double konv2(int d) {
      return d;
    }
    void vypis() {
      int k = konv1(4);
      double j = konv2((int)4.5);
      System.out.println("k = " + k + ", j = " + j);
    }
  }
  ```
  ```

Přetížené metody

- jsou metody, které mají stejná jména, ale různé hlavičky. Formální parametry se musí lišit počtem, typem nebo pořadím, příp. kombinací předchozích.
- Metodu nelze přetížit pouhou změnou návratové hodnoty.
- Kdykoliv je přetížená metoda volána, kompilátor vybere tu z metod, která přesně odpovídá počtu, typům a pořadí skutečných parametrů.
- Pozor na volání přetížených metod s využitím implicitní rozšiřující konverze. Zde je někdy potřeba při volání explicitně přetypovat skutečné parametry, aby typy přesně odpovídaly typům formálních parametrů.

Přetížení metod – příklad

```
class Pretizeni1 {
 int ctverec(int i) { return i * i; }
 double ctverec(double i) { return i * i; }
 //  long ctverec(int i) {...} // chyba
 long ctverec(long i) {
   return i * i;
 }
void vypis() {
   int j = ctverec(5);
   double d = ctverec(5.5);
   long l = ctverec(12345L);
   System.out.println("j = " + j + ", d = " + d
                      + ",   l = " + l);
 }
}
```

Parametry přetížených metod – příklad

- Máme přetíženou metodu s uvedenými formálními parametry:

```
1. void id(int a, double b) {...}
2. void id(int a, int b) {...}
3. void id(double a, int b) {...}
```

- Která varianta metody `id` bude spuštěna v případě volání s následujícími skutečnými parametry?

```
id(2,5);   id(2,5.3);    id(2,5.0);
id(2L,5);  id(2L,5.0);
```

- Máme přetíženou metodu s uvedenými formálními parametry:

```
void id(int a, double b) {...}
void id(double a, int b) {...}
```

- Která varianta metody `id` bude spuštěna v případě volání s následujícími skutečnými parametry?

```
id(2,5);   id(2,(double)5);
```

Umístění definic metod

- **Pokud voláme metody z téže třídy, mohou být tyto metody implementovány i za místem volání.**

```
void vypocet() {
    int i = 8;
    int j = naDruhou(i);

    ...
}

int naDruhou (int x) {
   return (i * i);
}
```

Samostudium!!!

Zde si před cvičením v následujícím týdnu nastudujte části 6.1, 6.2, 8.1 až 8.7, 9.3 až 9.13:

https://akela.mendelu.cz/~petrj/java-sbornik/toc.html