

# **Programování v jazyce Java**

---

## **Programování v jazyce Java**

---

---



---







# Obsah

1. Informace o charakteru, průběhu a hodnocení předmětu. Úvod do jazyka Java a prostředí BlueJ	1
Informace o průběhu, hodnocení, apod.	1
O předmětu...	1
Složky hodnocení předmětu	1
Kritéria hodnocení předmětu	2
Obsah (syllabus) předmětu	2
Cvičení	3
O přednášejícím...	3
Konzultační hodiny	3
Informační zdroje	4
Úvod do prostředí Java	4
Java jako programovací jazyk...	4
Z toho plyne, že...	5
Další charakteristiky	5
Java jako běhové prostředí	5
Java pro programátora (1)	5
Java pro programátora (2)	6
Hlavní domény Javy (1)	6
Hlavní domény Javy (2)	6
Javová platforma	6
Java je tedy dána...	7
Vývoj Javy	7
Specifikace a implementace Javy	7
Verze Javy - starší konvence	7
Verze Javy - konvence od verze 5.0 (1.5 postaru)	8
Aktuální verze	8
Co je nového v Javě 5.0	8
Licence k použití (a redistribuci) Javy	9
Stažení distribuce Sun	9
Typy distribucí Javy (Sun)	9
Obsah vývojové distribuce Javy	9
Obsah vývojové distribuce Javy (2)	10
Nástroje ve vývojové distribuci	10
Pomocné nástroje ve vývojové distribuci	11
První kroky javového programování v prostředí BlueJ	11
Struktura javového programu	11
Spuštění a běh javového programu	12
Prostředí BlueJ	12
Stažení BlueJ	12
Instalace BlueJ	13
Spuštění BlueJ	14
Nastavení českého prostředí BlueJ	15
Otevření hotového programu (projektu) v BlueJ	16

Spuštění hotového programu .....	17
Interaktivní vytvoření objektu (instance) .....	19
Práce s objektem (instancí) .....	21
Úprava hotového programu v BlueJ .....	24
Sestavení a spuštění programu bez BlueJ .....	25
Základní životní cyklus javového programu - bez BlueJ .....	26
Demo "Ahoj!" .....	26
Překlad a spuštění "Ahoj!" .....	27
Vytvoření zdrojového textu "Ahoj!" ("for dummies") .....	28
Překlad "Ahoj!" ("for dummies") .....	28
Spuštění "Ahoj!" ("for dummies") .....	28
Co znamená spustit program? .....	29
Praktické informace (aneb co je nutné udělat) .....	29
Praktické informace (aneb co je vhodné udělat) .....	29
Odkazy .....	30
2. Základní pojmy OOP. Třída, objekt, jeho vlastnosti. Metody, proměnné. Konstruktory. ....	31
Třída, objekt, jejich vlastnosti .....	31
Co je třída a objekt? .....	31
Vlastnosti objektu (1) .....	32
Vlastnosti objektu (2) .....	32
Deklarace a použití třídy .....	32
Příklad - deklarace třídy Clovek .....	32
Příklad - použití třídy Clovek (1) .....	33
Příklad - použití třídy Clovek (2) .....	34
Vytváření objektů .....	34
Shrnutí .....	34
Proměnné .....	34
Proměnné - deklarace .....	34
Proměnné - datový typ .....	35
Proměnné - jmenné konvence .....	35
Proměnné - použití .....	36
Proměnné - modifikátory přístupu .....	36
Proměnné - použití (2) .....	36
Metody .....	37
Metody .....	37
Metody - příklad .....	37
Volání metod .....	38
Volání metod - příklad .....	38
Parametry metod .....	38
Předávání skutečných parametrů metodám .....	39
Příklad předávání parametrů - primitivní typy .....	39
Příklad předávání parametrů - objektové typy (1) .....	40
Příklad předávání parametrů - objektové typy (2) .....	40
Návrat z metody .....	41
Konstruktory .....	41
Konstruktory .....	41
Konstruktory (2) .....	42
Přetěžování metod .....	42
Přetěžování .....	42

---













Přetěžování - příklad .....	43
Přetěžování - příklad (2) .....	43
Přetěžování - příklad (3) .....	43
Odkazy na objekty (instance) .....	44
Odkazy na objekty (instance) .....	44
Přiřazování objektových proměnných .....	44
Vracení odkazu na sebe .....	45
Řetězení volání .....	45
Statické proměnné a metody .....	46
Proměnné a metody třídy - statické .....	46
Příklad statické proměnné a metody .....	46
3. Struktura složitějších programů. Rozhraní. Dědičnost. ....	48
Objektové modelování reality .....	48
Kroky řešení reálného problému na počítači .....	48
Vývoj software je proces... ..	48
Celkový rámec vývoje SW .....	48
Metodiky vývoje SW .....	49
Metodika typu "vodopád" .....	49
Srovnání Java - Pascal .....	49
Organizace programových souborů .....	50
Organizace zdrojových souborů .....	50
Příklad - svět chovatelů a jejich psů .....	50
Shromáždění informací o realitě .....	50
Jak zachytíme tyto informace .....	51
Modelování reality pomocí tříd .....	51
Rozhraní .....	51
Rozhraní .....	51
Co je rozhraní .....	52
Deklarace rozhraní .....	52
Implementace rozhraní .....	52
Využití rozhraní .....	53
Dědičnost .....	53
.....	53
Dědičnost .....	53
Terminologie dědičnosti .....	54
Jak zapisujeme dědění .....	54
Dědičnost a vlastnosti tříd .....	54
Příklad .....	55
Příklad - co tam bylo nového .....	56
Přístupová práva (viditelnost) .....	56
Přístupová práva .....	56
Granularita omezení přístupu .....	56
Typy omezení přístupu .....	57
Kde jsou která omezení aplikovatelná? .....	57
Příklad - public  .....	57
Příklad - protected  .....	58
Příklad - přátelský .....	58
Příklad - private .....	59

Když si nevíte rady .....	59
Přístupová práva a umístění deklarací do souborů .....	60
Organizace tříd do balíků .....	60
Zápis třídy do zdrojového souboru .....	60
Organizace tříd do balíků .....	60
Balíky .....	61
Příslušnost třídy k balíku .....	61
Deklarace <code>import NázevTřídy</code>  .....	62
Deklarace <code>import názevbalíku.*</code>  .....	62
4. Datové typy: primitivní, objektové, pole. Výrazy. ....	63
Úvod k datovým typům v Javě .....	63
Primitivní vs. objektové datové typy - opakování .....	63
Přiřazení proměnné primitivního typu - opakování .....	63
Přiřazení objektové proměnné - opakování .....	63
Primitivní datové typy .....	64
Primitivní datové typy .....	64
Integrální typy - celočíselné .....	65
Integrální typy - "char" .....	65
Typ <code>char</code>  - kódování .....	66
Čísla s pohyblivou řádovou čárkou .....	66
Vestavěné konstanty s pohyblivou řádovou čárkou .....	67
Typ logických hodnot - <i>boolean</i> .....	67
Typ <code>void</code>  .....	67
Pole .....	67
Pole v Javě .....	68
Pole (2) .....	68
Pole - co když deklarujeme, ale nevytvoříme? .....	69
Pole - co když deklarujeme, vytvoříme, ale nenaplníme? .....	70
Kopírování polí .....	70
Operátory a výrazy .....	71
Aritmetické .....	71
Logické .....	71
Relační (porovnávací) .....	72
Bitové .....	72
Operátor podmíněného výrazu ? :  .....	73
Operátory typové konverze (přetypování) .....	73
Operátor zřetězení +  .....	73
Priority operátorů a vytváření výrazů .....	74
Porovnávání objektů .....	74
Relační (porovnávací) .....	74
Porovnávání objektů .....	75
Porovnávání objektů - příklad .....	75
Metoda <code>hashCode</code> .....	76
Metoda <code>hashCode</code> - příklad .....	76
5. Řídící struktury: větvení, cykly. ....	78
Příkazy v Javě .....	78
Příkazy v Javě .....	78

Přiřazení .....	78
Přiřazení v Javě .....	78
Přiřazení primitivní hodnoty .....	79
Přiřazení odkazu na objekt .....	79
Volání metod a návrat z nich .....	80
Volání metody .....	80
Návrat z metody .....	80
Řízení toku uvnitř metod - větvení, cykly .....	80
Řízení toku programu v těle metody .....	80
Cyklus s podmínkou na začátku .....	81
Doporučení k psaní cyklů/větvení .....	81
Příklad použití "while" cyklu .....	82
Cyklus s podmínkou na konci .....	82
Příklad použití "do-while" cyklu .....	82
Cyklus "for" .....	83
Příklad použití "for" cyklu .....	83
Doporučení k psaní <code>for</code> cyklů (1) .....	84
Doporučení k psaní <code>for</code> cyklů (2) .....	84
Vícecestné větvení "switch - case - default" .....	85
Vnořené větvení .....	85
Vnořené větvení (2) .....	86
Řetězené "if - else if - else" .....	86
Příkazy "break" .....	87
Příkaz "continue" .....	87
"break" a "continue" s návěštím .....	88
Doporučení k příkazům break a continue .....	88
6. Ladění programů, testování jednotek (junit) .....	90
Ladění programů (debugging) .....	90
Ladění programů v Javě .....	90
Ještě lepší... .....	90
Běhové ověření podmínky - assert .....	91
Postup při práci s <code>assert</code> .....	91
Ukázka použití <code>assert</code> (1) .....	91
Ukázka použití <code>assert</code> (2) .....	93
Ukázka použití <code>assert</code> (3) .....	93
Ukázka použití <code>assert</code> (4) .....	93
Testování jednotek nástrojem junit .....	93
Postup při práci s <code>JUnit</code> .....	94
Ukázka použití <code>JUnit</code> (1) .....	94
Ukázka použití <code>JUnit</code> (2) .....	95
Ukázka použití <code>JUnit</code> (3) .....	96
Ukázka použití <code>JUnit</code> (4) .....	97
Návrh dle kontraktu .....	98
Postup při práci s <code>jass</code> .....	98
Odkazy .....	99
7. Pokročilejší objektový návrh. dědičnost rozhraní, implementace více rozhraní. Abstraktní třídy. ....	101



Implementace více rozhraní současně .....	101
Implementace více rozhraní současně .....	101
Implementace více rozhraní současně - příklad .....	101
Rozšiřování rozhraní .....	102
Rozšiřování rozhraní .....	102
Rozšiřování rozhraní - příklad .....	102
Rozhraní - poznámky .....	103
Abstraktní třídy .....	103
Abstraktní třídy .....	103
Abstraktní třídy (2) .....	103
Příklad rozhraní - abstraktní třída - neabstraktní třída .....	104
8. Dynamické datové struktury (kontejnery). ....	105
Kontejnery, iterátory, kolekce .....	105
Kontejnery .....	105
Základní kategorie kontejnerů .....	106
Kontejnery - rozhraní, nepovinné metody .....	106
Kontejnery - souběžný přístup, výjimky .....	106
Iterátory .....	106
Kolekce .....	107
Seznamy .....	107
Seznamy .....	107
Seznamy a obyčejné iterátory - příklad .....	107
Iterátor po seznamu - příklad .....	109
Množiny .....	110
Množiny .....	110
Množina - příklad .....	110
Uspořádané množiny .....	111
Uspořádaná množina - příklad s chybou .....	112
Uspořádaná množina - příklad OK .....	113
Mapy .....	115
Mapy .....	115
Mapa - příklad .....	115
Uspořádané mapy .....	117
Uspořádaná mapa - příklad .....	117
Uspořádaná mapa - příklad s komparátorem .....	119
Starší typy kontejnerů .....	120
Historie .....	120
Srovnání implementací a odkazy .....	121
Srovnání implementací kontejnerů .....	121
Odkazy .....	121
9. Výjimky. ....	122
Výjimky .....	122
Co a k čemu jsou výjimky .....	122
Výjimky technicky .....	122
Syntaxe kódu s ošetřením výjimek .....	123
Syntaxe metody propouštějící výjimku .....	123
Reakce na výjimku .....	124
Kaskády výjimek .....	124
Kategorizace výjimek a dalších chybových objektů .....	124

Vlastní hierarchie výjimek .....	125
Klauzule <code>finally</code>  .....	125
Odkazy .....	125
10. Vstupy a výstupy v Javě. ....	127
Vstupy a výstupy v Javě .....	127
Koncepte vstupně/výstupních operací v Javě .....	127
Práce se soubory .....	127
Třída <code>File</code>  .....	128
Třída <code>File</code>  (2) .....	130
Třída <code>File</code>  (3) .....	131
Práce s adresáři .....	133
Práce s binárními proudy .....	133
Vstupní binární proudy .....	133
Důležité neabstraktní třídy odvozené od <code>InputStream</code>  .....	134
Další vstupní proudy .....	134
Práce se znakovými proudy .....	135
Výstupní proudy .....	135
Konverze: znakové <-> binární proudy .....	136
Serializace objektů .....	136
Odkazy .....	137
11. Pokročilejší témata. Použití javadoc. Distribuce aplikací - jar. Generické datové typy. Novinky v Javě 5. ....	138
Dokumentace aplikací .....	138
Dokumentace javových programů .....	138
Typy komentářů .....	138
Kde uvádíme dokumentační komentáře .....	139
Generování dokumentace .....	139
Značky javadoc  .....	140
Příklad zdrojového textu se značkami javadoc .....	140
Spouštění javadoc  .....	141
Příklady .....	141
Distribuce aplikací .....	142
Distribuce aplikací .....	142
Spuštění <code>jar</code>  .....	142
Volby <code>jar</code>  .....	143
<code>jar</code>  - příklad .....	144
Rozšíření <code>.jar</code>  archivů .....	146
Tvorba spustitelných archivů .....	146
Vytvoření spustitelného archivu - příklad .....	146
Spuštění archivu - příklad .....	147
Další příklad spuštění <code>jar</code>  .....	147
Generické typy .....	147
Generické datové typy (generics) .....	147
Základní syntaxe .....	148
Jednoduché využití v metodách .....	149

První příklad použití .....	150
Cyklus foreach .....	151
Žolíci (wildcards) .....	151
Generické metody .....	155
Generics metody vs. wildcards .....	155
Pole .....	156
Vícenásobná vazba generics .....	156
Závěr .....	157
Další změny v Javě 5 .....	158
Zrychlení startu aplikací .....	158
Výčtový typ .....	158
Ostatní vylepšení Javy 5 .....	159
12. Informace k testům. ....	160
Písemky .....	160
První písemka .....	160
Druhá písemka .....	160
Třetí (zkoušková) písemka .....	160
13. Dodatky - Co je nového v Javě 1.5. ....	162
Další změny v Javě 5 .....	162
Zrychlení startu aplikací .....	162
Výčtový typ .....	162
Ostatní vylepšení Javy 5 .....	163

---

## Seznam obrázků

1.1. Stažení BlueJ .....	12
1.2. Spuštění BlueJ .....	14
1.3. BlueJ spuštěno .....	14
1.4. BlueJ v češtině .....	15
1.5. Otevření hotového projektu v BlueJ .....	16
1.6. Spuštění hotového projektu v BlueJ .....	17
1.7. Parametry spuštění .....	18
1.8. Výsledek spuštění na konzole BlueJ .....	19
1.9. Editace zdrojového kódu třídy Hello .....	20
1.10. Specifikace jména odkazu na novou instanci Hello .....	20
1.11. Vytvořená instance třídy Hello .....	21
1.12. Volání (spuštění) metody go() .....	22
1.13. Výsledek spuštění go() .....	23
1.14. Metoda go() upravena a třída úspěšně přeložena .....	24
1.15. Výsledek běhu upravené třídy Hello .....	25
4.1. Dva lidi jsou stejní, mají-li stejná příjmení .....	75
4.2. Třída Clovek s metodami equals a hashCode .....	76
6.1. Třídy AssertDemo, Zlomek .....	92
6.2. Správný postup překladu AssertDemo .....	93
6.3. Spuštění AssertDemo s povolením assert .....	93
6.4. Spuštění AssertDemo bez povolení assert .....	93
6.5. Upravená třída Zlomek s porovnáním a součtem .....	94
6.6. Testovací třída JUnitDemo .....	95
6.7. Spouštění testovače nad testovací třídou JUnitDemo .....	96
6.8. Testovač spouštějící třídu JUnitDemo .....	97
6.9. Testovací třída JUnitDemo našla chybu .....	97
8.1. Pohyb po seznamu iterátorem .....	107
8.2. Spuštění pg. s iterátorem .....	108
8.3. Pohyb seznamovým iterátorem .....	109
8.4. Spuštění pg. se seznamovým iterátorem .....	109
8.5. Vložení prvků do množiny a dotaz na přítomnost .....	110
8.6. Spuštění pg. s množinou .....	111
8.7. Vložení neporovnatelných prvků do uspořádané množiny .....	112
8.8. Spuštění nefunkčního pg. s uspořádanou množinou .....	113
8.9. Vložení porovnatelných prvků do uspořádané množiny .....	114
8.10. Spuštění funkčního pg. s uspořádanou množinou .....	114
8.11. Vložení lidí do mapy pod klíčem příjmení a vyhledání člověka .....	115
8.12. Spuštění funkčního pg. s mapou .....	116
8.13. Vložení lidí do mapy pod uspořádaným klíčem příjmení a projde je .....	117
8.14. Spuštění funkčního pg. s uspořádanou mapou .....	118
8.15. Vložení účtů do mapy pod uspořádaným klíčem člověka - vlastníka .....	119
8.16. Spuštění funkčního pg. s uspořádanou mapou .....	120
11.1. Volby nástroje JAR .....	143
11.2. Třída JarDemo .....	144
11.3. Vytvoření archívu se všemi soubory z podadresáře tomp/ucebnice/jar .....	145

11.4. .jar archiv v okně PowerArchiveru .....	145
11.5. Vybalení všech souborů z archívu .....	145
11.6. Soubor manifestu .....	146
11.7. Zabalení archívu s manifestem .....	146
11.8. Spuštění aplikace z archívu .....	147

---

## Seznam příkladů

3.1. Příklad kompletního zdrojového kódu třídy .....	55
--	----

---

# Kapitola 1. Informace o charakteru, průběhu a hodnocení předmětu. Úvod do jazyka Java a prostředí BlueJ

## Informace o průběhu, hodnocení, apod.

- Souvislosti, prerekvizity, návaznosti
- Hodnocení a jeho složky
- Kontakty a konzultační hodiny
- Odkazy na informační zdroje

## O předmětu...

Prakticky zaměřený bakalářský předmět

Cílem je naučit základním principům objektového programování a algoritmizace

Souvisí s

- IB001 - **Úvod do programování** (předpokládají se znalosti na úrovni IB001)
- IB002 - **Návrh algoritmů I** (v PB162 se prakticky implementují vybrané algoritmy probírané v IB002)

Předpokládají se základní znalosti strukturované algoritmizace a programování (v rozsahu Úvodu do programování), tj. např.:

- základní příkazy, sestavování jednoduchých výrazů
- základní datové typy (celá a reálná čísla, logické proměnné, řetězce)
- základní řídicí struktury - větvení, cykly, procedury/funkce

## Složky hodnocení předmětu

Hodnocení má tři složky:

- **40 bodů - hodnocení úloh** řešených samostatně v průběhu semestru (na cvičeních, ve volném čase...)
- **12 bodů - první písemka** řešení jednoduché praktické úlohy přímo u počítače, v době cvičení, zhruba po prvním měsíci až 5 týdnech výuky (říjen). Čas na písemku: i se zadáním během hodinového cvičení. Bližší info viz „První písemka“.
- **18 bodů - druhá písemka** řešení praktické úlohy přímo u počítače, v době cvičení, na konci výuky v semestru (prosinec). Bližší info viz „Druhá písemka“.
- **30 bodů - třetí písemka** řešení rozsáhlejší praktické úlohy přímo u počítače, ve zkouškovém období. Čas na písemku: delší než u prvních dvou. Bližší info viz „Třetí (zkoušková) písemka“.
- **Celkem 100 bodů**



### Poznámka

písemky v průběhu semestru (tj. první dvě) i úlohy hodnotí cvičící



### Poznámka

docházka do cvičení je povinná, ale netvoří součást bodového hodnocení

## Kritéria hodnocení předmětu

Doporučené ukončení je zkouškou. K tomu potřebujete:

- A 94 - 100 bodů
- B 88 - 93 bodů
- C 82 - 87 bodů
- D 76 - 81 bodů
- E 70 - 75 bodů
- F 0 - 69 bodů

K ukončení zápočtem postačí:

- 60 bodů

## Obsah (syllabus) předmětu



- viz aktuální informace o předmětu PB162  
[<http://is.muni.cz/predmety/predmet.pl?jazyk=cs;id=232608>] na IS MU

## Cvičení

Cvičení jsou dvouhodinová, konají se pod vedením příslušných kvalifikovaných cvičících v počítačových učebnách, zpravidla B116, B117 - viz váš rozvrh.

### Náplň

- Hlavním obsahem je *konzultovaná samostatná práce* na bodovaných úlohách.
- cvičení jsou *dvouhodinová*, účast na přednáškách je však přesto žádoucí
- předpokládá se i jistý netriviální podíl *práce mimo cvičení*
- odměnou vám, kromě získaných znalostí a dovedností, budou *celkem 4 kr. + za ukončení*

## O přednášejícím...

Tomáš Pitner

- kanc. B307 (3. patro budovy B)
- tel. 54949 5940 (z tlf. mimo budovu), kl. 5940 (volání v rámci fakulty i celé MU)
- e-mail: [tomp@fi.muni.cz](mailto:tomp@fi.muni.cz) [<mailto:tomp@fi.muni.cz>]
- www TP: <http://www.fi.muni.cz/~tomp>

materiály PB162-Java (IS): <https://is.muni.cz/auth/el/1433/podzim2005/PB162/>  
[<https://is.muni.cz/auth/el/1433/podzim2005/PB162/>]

## Konzultační hodiny

Primárním konzultačním bodem jsou vaši cvičící. Cvičení jsou vedena mj. právě z důvodu možnosti konzultací.

Konzultace přímo s přednášejícím: vždy v kanc. B307, prosím o dodržování časů:

- **Po 9.00 - 11.00 (preferovaná doba)**
- **Čt 13.00 - 14.00**
- nebo jindy, dle dohody

## Informační zdroje

- Studijní materiály předmětu: <https://is.muni.cz/auth/el/1433/podzim2005/PB162/> (budou vystavovány postupně, pro celkový přehled o obsahu lze použít materiály loňské)
- Další zdroje, materiály z minulých let: <http://www.fi.muni.cz/~tomp/java>
- Knihy
  - TP: **Java - začínáme programovat**, Grada Publishing [<http://www.gradapublishing.cz>], 2002, <http://www.fi.muni.cz/~tomp/java/ucebnice>
  - Pavel Herout: **Učebnice jazyka Java**, Kopp [<http://www.kopp.cz>], 2000
  - (Pavel Herout: **Java - grafické uživatelské rozhraní a čeština**, Kopp [<http://www.kopp.cz>], 2001) - pro pokročilé
  - Rudolf Pecinovský: **Myslíme v jazyce Java 5.0**, Grada Publishing [<http://www.gradapublishing.cz>], 2005
  - Bruce Eckel: **Myslíme v jazyce Java - příručka programátora**, Grada Publishing [<http://www.gradapublishing.cz>], 2000
  - (Bruce Eckel: **Myslíme v jazyce Java - příručka zkušeného programátora**, Grada Publishing [<http://www.gradapublishing.cz>], 2000) - pro pokročilé
  - a další, viz např. <http://www.vltava.cz>
  - Joshua Bloch: **Java efektivně 57 zásad softwarového experta**, Grada Publishing [<http://www.gradapublishing.cz>]
  - Bogdan Kiszka: **1001 tipů a triků pro programování v jazyce Java**, Computer Press, 2003, informace na Vltavě [<http://www.vltava.cz/Store/GoodsDetail.asp?c=BogdanKiszka&sCGoodsID=SE00679120>]

## Úvod do prostředí Java

- Úvod, srovnání s jinými, oblasti použití Javy
- Distribuce, instalace, použití Java SDK

## Java jako programovací jazyk...

- jazyk "3. generace - 3GL" (imperativní jazyk vysoké úrovně)

- *univerzální* (není určen výhradně pro specifickou aplikační oblast)
- *objektově-orientovaný* (výpočet je realizován jako volání metod/zasílání zpráv objektů)
- ideovým předchůdcem je C++ (a evt. Delphi) (C++ zbaveno zbytečností a nepříjemností)
- *jednodušší než C++*
- reálným soupeřem je (Microsoft) C# (zatím převážně na platf. Windows)

## Z toho plyne, že...

- co se naučíme v Javě, v C# jako když najdeme...
- ale teď vážně: *Java podporuje vytváření správných návyků v objektovém programování*
- a naopak systematicky brání *přenosu některých špatných návyků z jiných jazyků*

## Další charakteristiky

- program v Javě je *meziplatformně přenositelný* na úrovni *zdrojového i přeloženého kódu*
- je to umožněno tím, že přeložený javový program běží v tzv. *Java Virtual Machine* [<http://java.sun.com/docs/books/vmspec/>] (JVM)
- zdrojový i přeložený kód je tedy přenositelný mezi všemi obvyklými platformami (UNIX, Windows, MAC OS X, ale také sálové počítače, minipočítače typu IBM AS/400 apod.)
- tedy všude tam, kde *existuje příslušná JVM*

## Java jako běhové prostředí

Kód je při běhu dobře zabezpečen:

- je možné nastavit úroveň přístupu k hostitelskému systému pomocí tzv. Security Manageru [<http://www.securingsjava.com/chapter-two/chapter-two-8.html>]
- je možné ověřovat před spuštěním elektronický podpis kódu

## Java pro programátora (1)

- jazyk vhodný pro *efektivní (rychlé) psaní přehledných programů* (mj. také díky *dokumentačním možnostem*)

- v průměru *vyšší produktivita* programátorské práce v Javě než v C++
- dnes již stejně aktivních programátorů v Javě jako v C++
- zdarma dostupné nezměrné množství knihoven pro různorodé aplikační oblasti, např. na SourceForge [<http://java.foundries.sourceforge.net/>] a tisících dalších místech
- k dispozici je řada kvalitních vývojových prostředí (i zdarma) - NetBeans [<http://www.netbeans.org>], JBuilder [<http://www.borland.com>], Visual Age for Java [<http://www.ibm.com>], Eclipse [<http://eclipse.org>], IDEA [<http://www.intelij.com/>]

## Java pro programátora (2)

Konkrétní možnosti:

- v Javě se dobře píše *vícevláknové aplikace* (multithreaded applications)
- Java má automatické *odklizení nepoužitelných objektů* (automatic garbage collection)
- Java je jednodušší než C++ (méně syntaktických konstrukcí, méně nejednoznačností v návrhu)

## Hlavní domény Javy (1)


- Škálovatelné výkonné *aplikace běžící na serverech* (Java Enterprise Edition [<http://java.sun.com/j2ee/>])
- Aplikace na *přenosných a vestavěných zařízeních* (Java Micro Edition [<http://java.sun.com/j2me/>])
- Přenositelné *desktopové i grafické/okénkové (GUI) aplikace*
- *Výukové účely* (nahrazuje Pascal jako referenční jazyk)
- Další přenositelné aplikace

## Hlavní domény Javy (2)

- Webové aplikace (servlety, JSP - konkurence proprietárním ASP, SSI, ... a pomalým CGI)
- Zpracování semistrukturovaných dat (XML)
- Aplikace distribuované po síti (applety nebo Java Web Start)

## Javová platforma

Javovou platformu tvoří:

- **Java Virtual Machine**
- **Java Core API** (základní knihovna tříd)
- **překladač** (přístupný např. příkazem `javac`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?javac>]) a další vývojové nástroje

## Java je tedy dána...

- **definicí jazyka** (Java Language Definition) - syntaxe a sémantika jazyka
- **popisem chování JVM**
- **popisem Java Core API**

## Vývoj Javy

- *nejrychleji* se vyvíjí Java Core API
- chování JVM se mění např. pokud se objeví bezpečnostní "díra" nebo nelze-li dosáhnout požadované změny chování pomocí modifikace Java Core API
- daleko konzervativnější je samotný *jazyk* - *mění se zřídka*, ale přece: např. Java2, v1.4 přidává nové klíčové slovo *assert*, Java 5.0 (postaru 1.5) obohacuje jazyk o *enum* a další.

## Specifikace a implementace Javy

- **Specifikace** Javy (tzv. "Editions") - např.: *Java 2 Standard Edition, 1.4* nebo *Java 2 Standard Edition 5.0*
- **Implementace** Javy ("Development Kits" nebo "Runtime Environments") - např.: *Java 2 Software Development Kit, 5.0* - obsahuje vývojové nástroje
- *Java 2 Runtime Enviroment, 5.0* - obsahuje jen **běžové prostředí** pro spouštění hotových přeložených pg.

## Verze Javy - starší konvence

- Verze Javy byly až do verze 1.5 (ponovu 5.0) děleny na "*Java* (před Java 2)" a "*Java 2*"

- číslování verzí:
  - tzv. major číslo, např. Java 2, v**1.4**
  - tzv. minor číslo, např. Java 2, v1.4.**2**
- změnu minor (třetího) čísla doprovází jen odstraňování chyb
- při změně major (druhého) čísla se může měnit Core API a někdy i jazyk
- ke změně prvního čísla dochází až teď, s verzí 5.0 (postaru 1.5) - tj. s celkovou změnou pojmenovovacího schématu

## Verze Javy - konvence od verze 5.0 (1.5 postaru)

Nové schéma číslování verzí počínaje 5.0 (postaru 1.5):

- tzv. major číslo, např. Java 2, **5.0**
- tzv. minor číslo, např. Java 2, 5.**0**

## Aktuální verze

Stav k září 2005:

- Java 2 Standard Edition v5.0 je stabilní verzí pro všechny platformy
- Java 2 Enterprise Edition v1.4 je stabilní verzí pro všechny platformy
- Připravována je Java 2 Enterprise Edition v5.0
- aktuální informace najdete vždy na webu <http://java.sun.com>

## Co je nového v Javě 5.0

Generics	generika (generické datové typy) technika umožňující silnou typovou kontrolu v době překladu - např. v metodách manipulujících s prvky dynamických datových struktur
Enhanced for Loop	umožňuje snadnou a typově bezpečnou iteraci po prvcích kolekcí a polí
Autoboxing/Unboxing	zajišťuje automatickou konverzi mezi primitivními typy a příslušnými objektovými "obalujícími" typy
Typesafe Enums	typově bezpečné výčtové typy

Varargs	umožňují specifikovat proměnný počet parametrů metod
Static Import	dovoluje odkazovat se na statické prvky bez uvádění jména třídy (rozhraní)
Metadata (Annotations)	dovolují specifikovat u tříd, rozhraní, metod metadata (anotace) dostupná i za běhu

## Licence k použití (a redistribuci) Javy

- používání Javy pro běžný vývoj (i komerční) je zdarma
- redistribuce javového **vývojového prostředí** je dovolena pouze s **licencí** od Sunu
- redistribuce javového **běžového prostředí** je možná **zdarma**
- distribuce vyvíjí Sun Microsystems Inc. (Javasoft) i další výrobci (např. IBM) a tvůrci Open Source

## Stažení distribuce Sun





- [java.sun.com](http://java.sun.com) [<http://java.sun.com>] (pro Windows, Solaris, Linux)
- lze stáhnout jak samotné vývojové prostředí (JDK), jen běžové prostředí (JRE) nebo JDK v balíčku s IDE (Integrated Development Environment) NetBeans [<http://netbeans.org>].
- dokumentace se stahuje z téhož místa, ale *samostatně* (nebo lze číst z WWW)
- celkově *vývojové prostředí J2SDK 1.4.2 vč. dokumentace* zabere cca **220 MB** na disku
- potřebná *velikost operační paměti* - min 64 MB, doporučeno **128 MB** (i více :-))

## Typy distribucí Javy (Sun)




Lze stáhnout:

- samotné vývojové prostředí (JDK),
- jen běžové prostředí (JRE),
- JDK v balíčku s grafickým (okénkovým) integrovaným vývojovým prostředím (IDE, Integrated Development Environment) NetBeans [<http://netbeans.org>].



## Obsah vývojové distribuce Javy





- **Vývojové nástroje (Development Tools)** v `bin`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?bin>] -- určené k vývoji, spouštění, ladění a dokumentování programů v Javě.
- **Běhové prostředí Javy (Java Runtime Environment)** se nalézá v `jre`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?jre>]. Obsahuje Java Virtual Machine (JVM), knihovnu tříd Java Core API a další soubory potřebné pro běh programů v Javě.
- **Přídavné knihovny (Additional libraries)** v podadresáři `lib`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?lib>] jsou další knihovny nutné pro běh vývojových nástrojů.
- **Ukázkové applety a aplikace (Demo Applets and Applications)** v `demo`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?demo>]. Příklady zahrnují i zdrojový kód.

## Obsah vývojové distribuce Javy (2)

- **Hlavičkové soubory pro C (C header Files)** - v `include`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?include>] - představují podporu pro psaní tzv. nativních metod přímo v jazyce C.
- **Staré hlavičkové soubory (Old Native Interface Headers)** - totéž, ale pro starší verzi rozhraní.
- **Zdrojový kód (Source Code)** knihoven z Java Core API se nalézá v archivu `src.zip`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?src.zip>].
- **Dokumentace (Documentation)** - v podadresáři `docs`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?docs>] - obsahuje dokumentaci k dané verzi JDK, k API, nejrůznější průvodce vývojem, dokumentaci k nástrojům, ukázkové programy a odkazy na související dokumentaci.

## Nástroje ve vývojové distribuci



Pod Windows jsou to `.exe`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.exe>] soubory umístěné v podadresáři `bin`  [<http://www.instantweb.com/foldoc/foldoc.cgi?bin>]

- **java**  [<http://www.instantweb.com/foldoc/foldoc.cgi?java>] - spouštěč (přeloženého bajtkódu)
- **javac**  [<http://www.instantweb.com/foldoc/foldoc.cgi?javac>] - překladač (`.java`    
[<http://www.instantweb.com/foldoc/foldoc.cgi?.java>]  $\rightarrow$  `.class`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?.class>])



- **javadoc**  [<http://www.instantweb.com/foldoc/foldoc.cgi?javadoc>] - generátor dokumentace API
- **jar**  [<http://www.instantweb.com/foldoc/foldoc.cgi?jar>] - správce archivů JAR (sbalení, rozbalení, výpis)
- **jdb**  [<http://www.instantweb.com/foldoc/foldoc.cgi?jdb>] - debugger
- **appletviewer**  [<http://www.instantweb.com/foldoc/foldoc.cgi?appletviewer>] - referenční prostředí pro spouštění appletů
- **javaws**  [<http://www.instantweb.com/foldoc/foldoc.cgi?javaws>] - referenční prostředí pro spouštění aplikací typu "Java Web Start" prostřednictvím Java Network Launching Protocol (JNLP)

## Pomocné nástroje ve vývojové distribuci

- **javah**  [<http://www.instantweb.com/foldoc/foldoc.cgi?javah>] - generátor hlavičkových souborů pro C - používá se při programování tzv. nativních (platformově závislých) metod dostupných přes *Java Native Interface* (JNI)
- **javap**  [<http://www.instantweb.com/foldoc/foldoc.cgi?javap>] - disassembler bajtkódu (např. pro ruční optimalizace, hledání chyb)

## První kroky javového programování v prostředí BlueJ

1. Seznámíme se se strukturou programu v Javě,
2. naučíme se spouštět hotové jednoduché programy ve výukovém prostředí BlueJ
3. a tyto programy upravovat.

## Struktura javového programu

- Každý netriviální javový program sestává z *více tříd* (class)
- Třídy jsou členěny do *balíků* (package)
- U běžné "desktopové" aplikace představuje vždy jedna (evt. více) tříd *vstupní bod* do programu - je to třída/y obsahující metodu *main*.

## Spuštění a běh javového programu

1. Spuštění programu se zahajuje vstupem do metody main.
2. Běh programu spočívá ve vytváření objektů a interakci s nimi, tj. volání jejich metod.
3. Běh končí, jakmile jsou provedeny všechny příkazy aktivované metodou main.

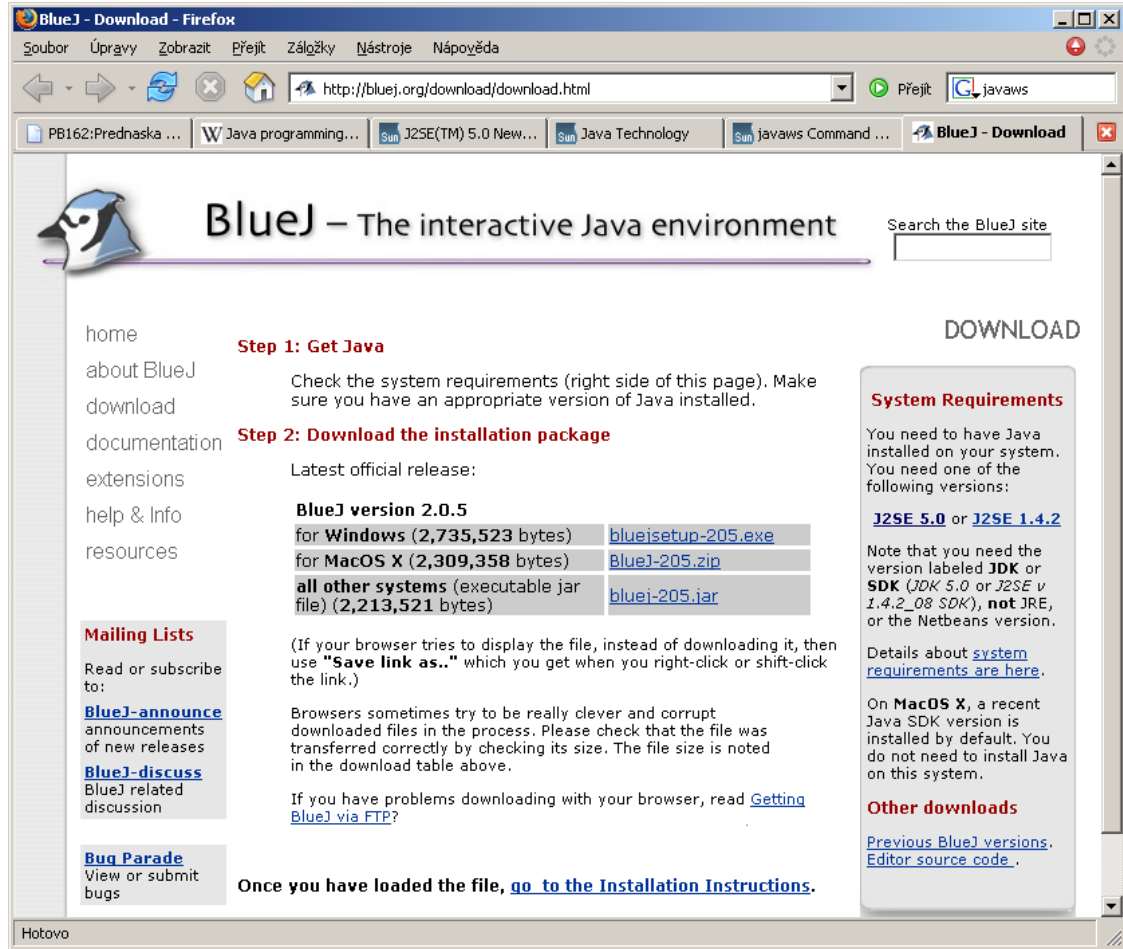
## Prostředí BlueJ

- Pro výuku v kurzu PB162 budeme převážně používat jednoduché, pro výuku určené, vývojové prostředí BlueJ.
- BlueJ je pro všechny platformy (Win, Linux...) zdarma dostupné na <http://bluej.org>.
- BlueJ je lokalizované pro češtinu vč. základního manuálu.
- Pro BlueJ existuje i jednoduchý animátor běhu programů - vizualizuje spouštění a chod programů.
- Toto prostředí je již předinstalované v síti FI.

## Stažení BlueJ

- BlueJ je již v síti FI instalováno, pro vlastní potřebu jej lze zdarma stáhnout z <http://bluej.org>.
- Předtím je třeba mít korektně nainstalováno samotné vývojové prostředí Java.

### Obrázek 1.1. Stažení BlueJ



## Instalace BlueJ

Po stažení příslušného instalačního balíčku postupujte v závislosti na platformě:

Windows poklepat (double-click) na staženém programu `bluejsetup-verze.exe` [G]  
[\[http://www.instantweb.com/foldoc/foldoc.cgi?bluejsetup-205.exe\]](http://www.instantweb.com/foldoc/foldoc.cgi?bluejsetup-205.exe) (např. `bluejsetup-205.exe`) [G] [\[http://www.instantweb.com/foldoc/foldoc.cgi?bluejsetup-205.exe\]](http://www.instantweb.com/foldoc/foldoc.cgi?bluejsetup-205.exe))

následovat pokyny instalačního programu

instalační program přitom nechá vybrat z předem instalovaných běhových prostředí Javy

Unix na konzole (v prostředí shellu) se přepněte (pomocí `cd`) [G]  
[\[http://www.instantweb.com/foldoc/foldoc.cgi?cd\]](http://www.instantweb.com/foldoc/foldoc.cgi?cd)) do adresáře se staženým balíčkem,  
 např. `bluej-205.jar` [G] [\[http://www.instantweb.com/foldoc/foldoc.cgi?bluej-205.jar\]](http://www.instantweb.com/foldoc/foldoc.cgi?bluej-205.jar)


spustíte instalaci příkazem `java -jar bluej-205.jar` [G]  
[\[http://www.instantweb.com/foldoc/foldoc.cgi?java-jar bluej-205.jar\]](http://www.instantweb.com/foldoc/foldoc.cgi?java-jar-bluej-205.jar)

při instalaci vyberte vhodný cílový adresář a zvolte z předem instalovaných běhových prostředí Javy

## Spuštění BlueJ

Po úspěšné instalaci na:

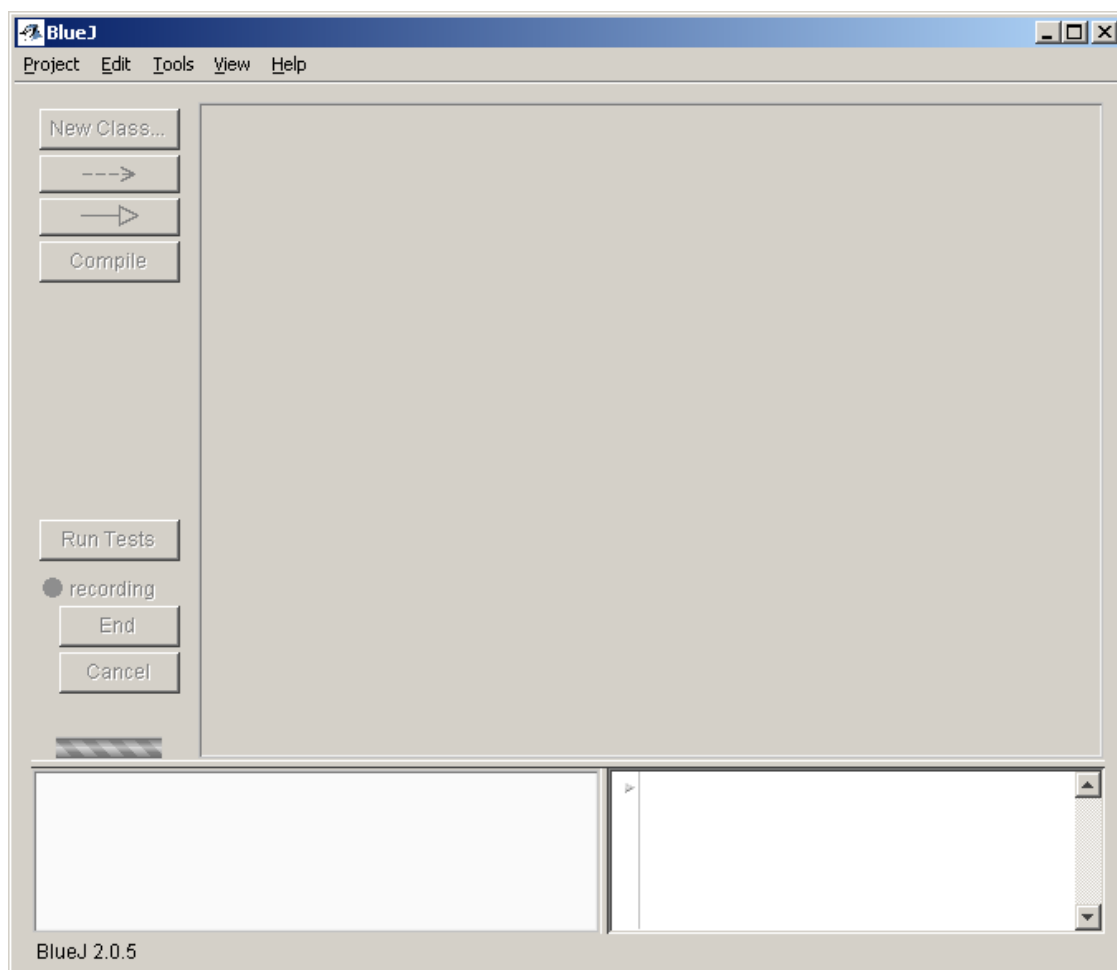
Windows    najdete v menu Start systému Windows složku *BlueJ* a zde spouštěcí odkaz *BlueJ* s ikonou ptáka

Unix        v adresáři, kam se BlueJ nainstalovalo, najdete spustitelný soubor `bluej`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?bluej>] - tím lze prostředí spustit.

**Obrázek 1.2. Spuštění BlueJ**



**Obrázek 1.3. BlueJ spuštěno**

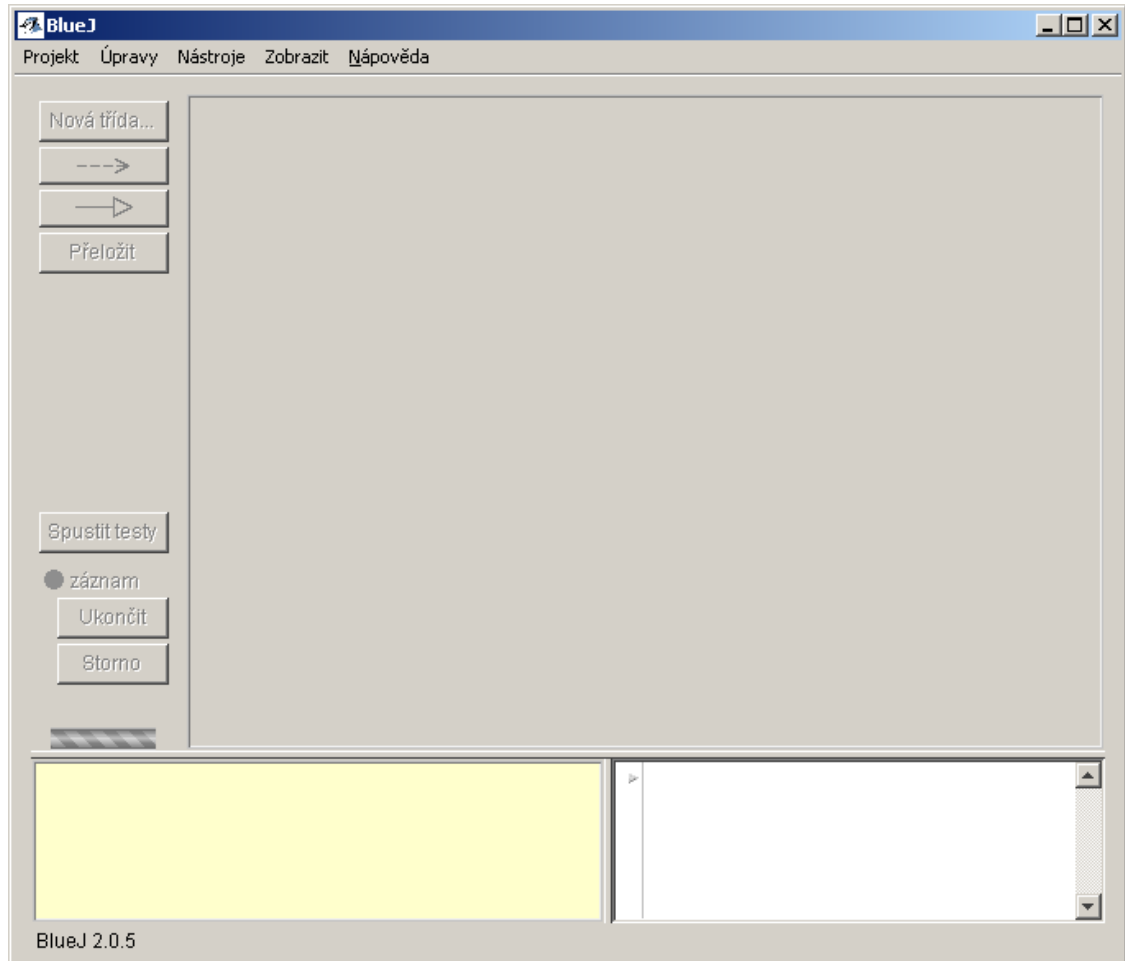


## Nastavení českého prostředí BlueJ



Chcete-li, aby nabídky (menu) a hlášky BlueJ byly v češtině, postupuje podle pokynů v Konfigurační doplněk pro prostředí BlueJ [<http://vyuka.pecinovsky.cz/mojj50/index.html#BlueJConfig>] (autor Rudolf Pecinovský).

Takto vypadá počeštěné prostředí:

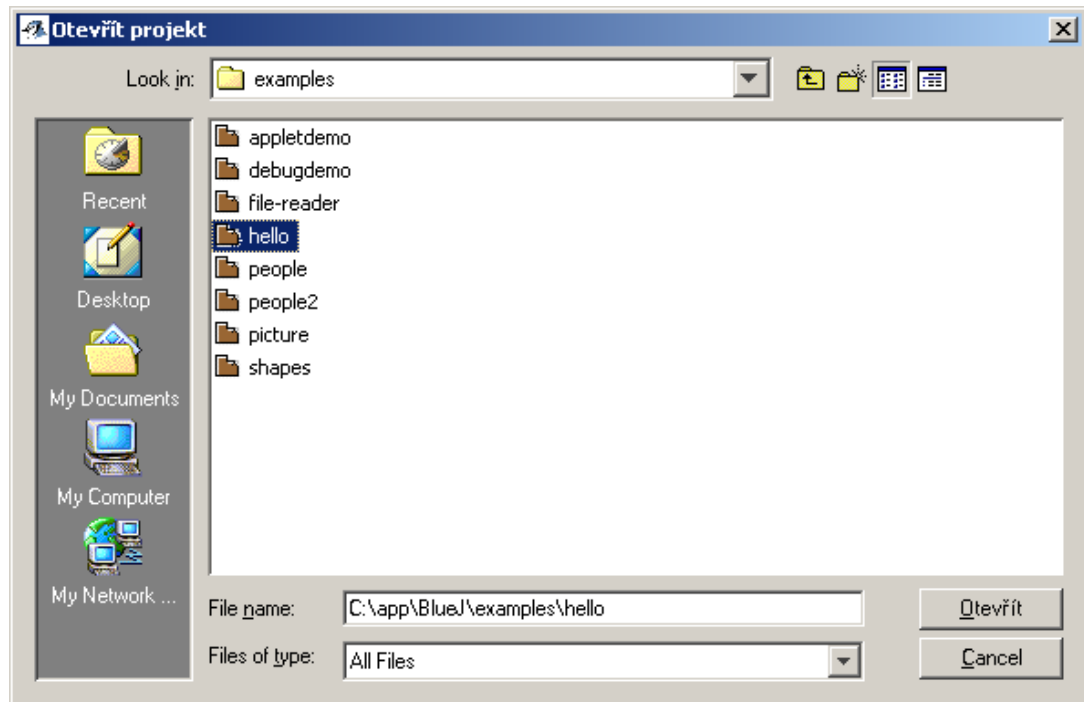
### Obrázek 1.4. BlueJ v češtině



## Otevření hotového programu (projektu) v BlueJ

- Každý javový program vytvořený v BlueJ představuje jeden *projekt*. Toto je specificky vlastnost BlueJ, ne Javy jako takové.
- Práce s hotovým programem v BlueJ tedy znamená *otevřít projekt* -> menu **Projekt**   
[<http://www.instantweb.com/foldoc/foldoc.cgi?Projekt>]/**Otevřít**  
[<http://www.instantweb.com/foldoc/foldoc.cgi?Otevřít projekt>]. **projekt** 

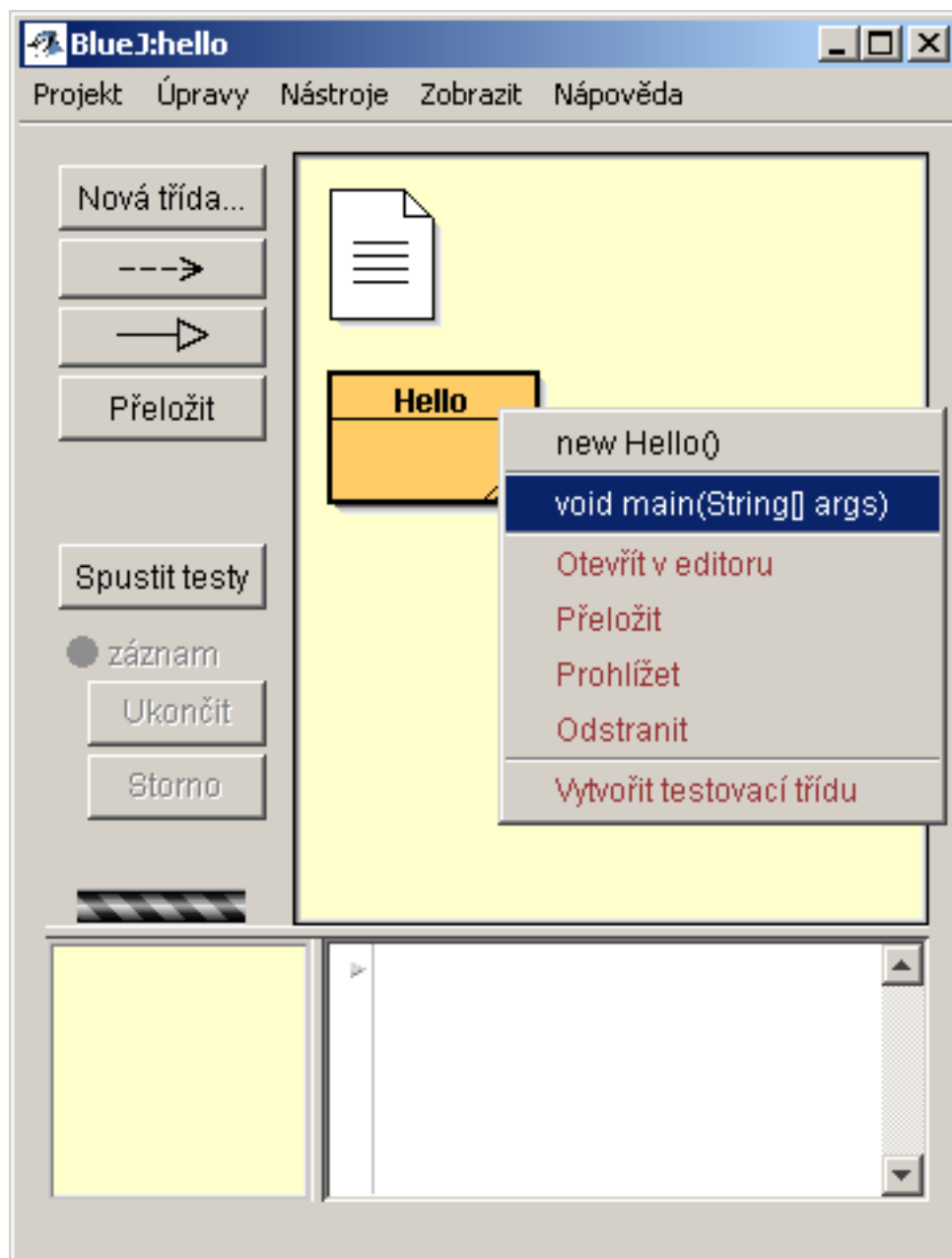
**Obrázek 1.5. Otevření hotového projektu v BlueJ**



## Spuštění hotového programu

- Každý běžný javový program spouštíme (nejen v BlueJ) aktivací metody `main`.
- V BlueJ to znamená kliknout pravým tlačítkem myši na ikoně příslušné třídy a vybrat `void main(String[] args)`.

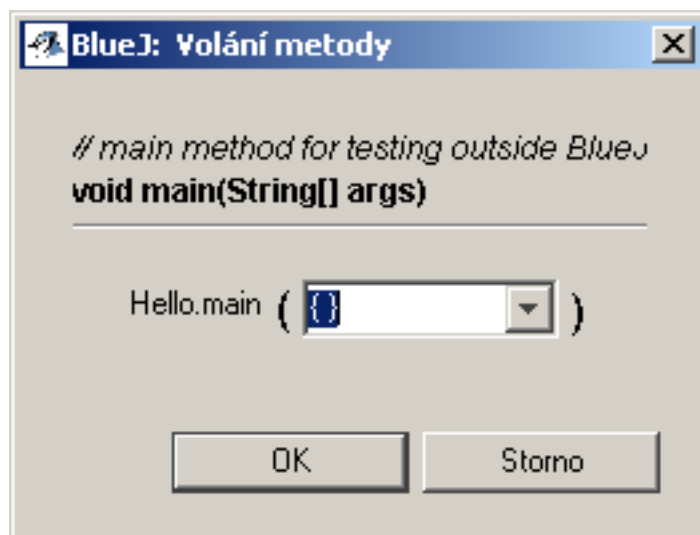
**Obrázek 1.6. Spuštění hotového projektu v BlueJ**



- Při spuštění uvádíme parametry, které metoda main dostane. Jelikož zatím žádné specifikovat nepotřebujeme, ponecháme dialog, jak je a stiskneme Enter (klikneme OK).

**Obrázek 1.7. Parametry spuštění**





- Výsledek spuštění programu se projeví v tzv. okně terminálu (konzole) BlueJ a vypadá takto:

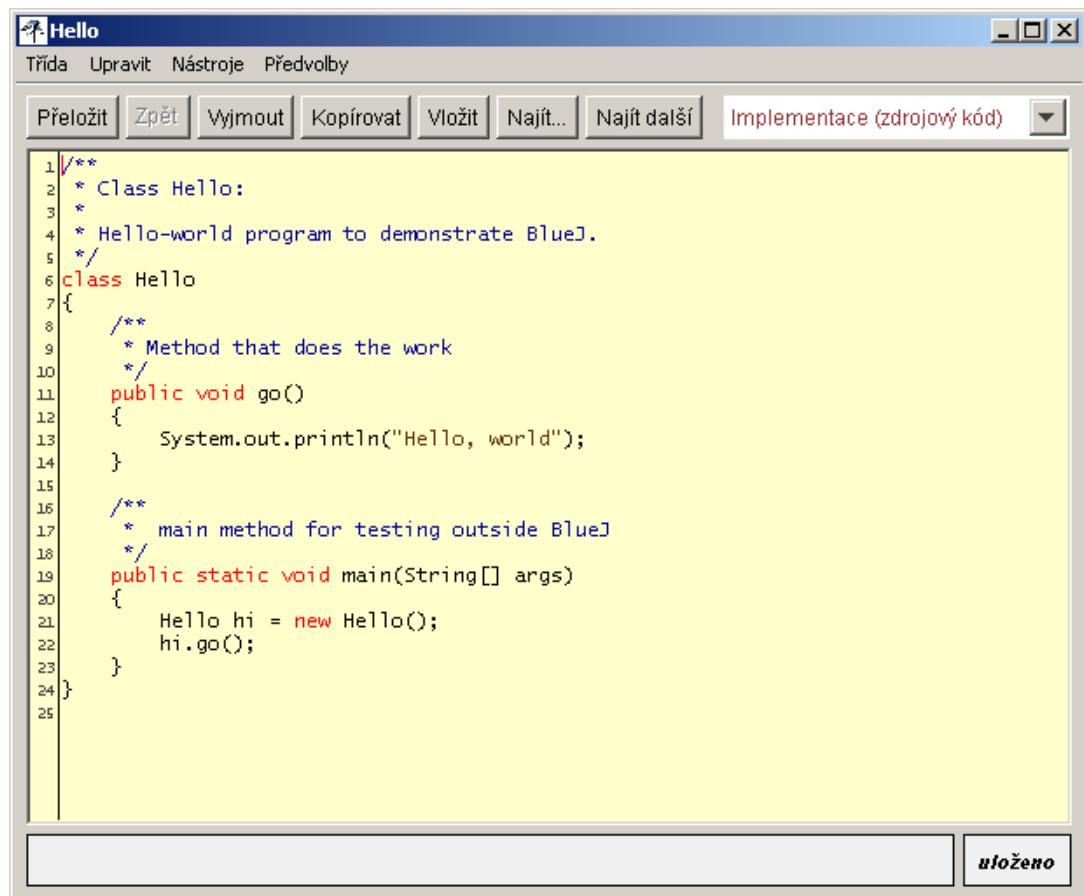
**Obrázek 1.8. Výsledek spuštění na konzole BlueJ**



## Interaktivní vytvoření objektu (instance)

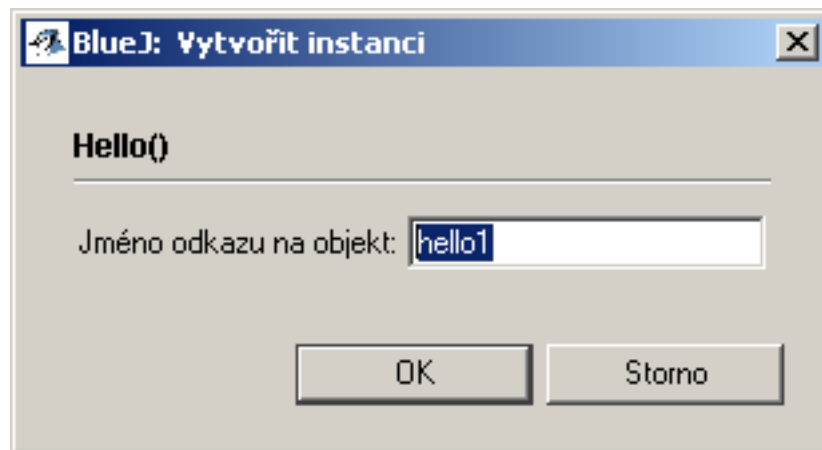
- Objektový přístup znamená především to, že program za běhu vytváří objekty a volá jejich metody. To lze v BlueJ snadno realizovat.
- V demostračním programu Hello máme jednu třídu (Hello), od níž lze vytvořit objekt (instanci) a na ní volat metody.
- Zdrojový kód třídy Hello (dostupný poklepnutím na její ikonu) prozradí, které metody zde máme k dispozici.

**Obrázek 1.9. Editace zdrojového kódu třídy Hello**



- Dále postupujeme:
  1. Pravým tlačítkem klikneme na ikonu Hello a vybereme *new Hello()*.
  2. do dialogu uvedeme název odkazu na vytvářený objekt typu Hello - můžeme ponechat *hello1*.

**Obrázek 1.10. Specifikace jména odkazu na novou instanci Hello**

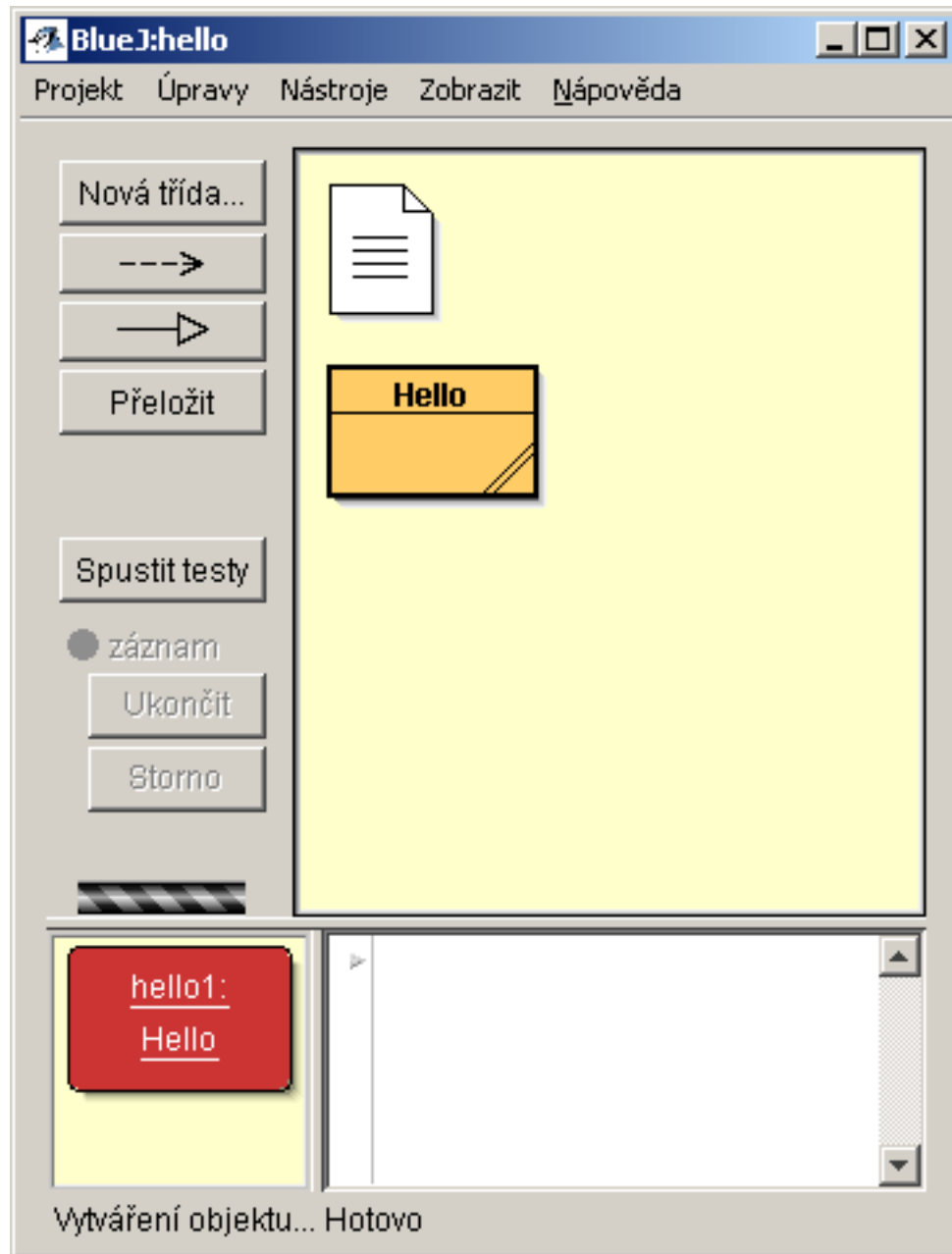


3. Vytvořený objekt (instance třídy `Hello`) je k dispozici jako červená krabička vlevo dole.

## Práce s objektem (instancí)

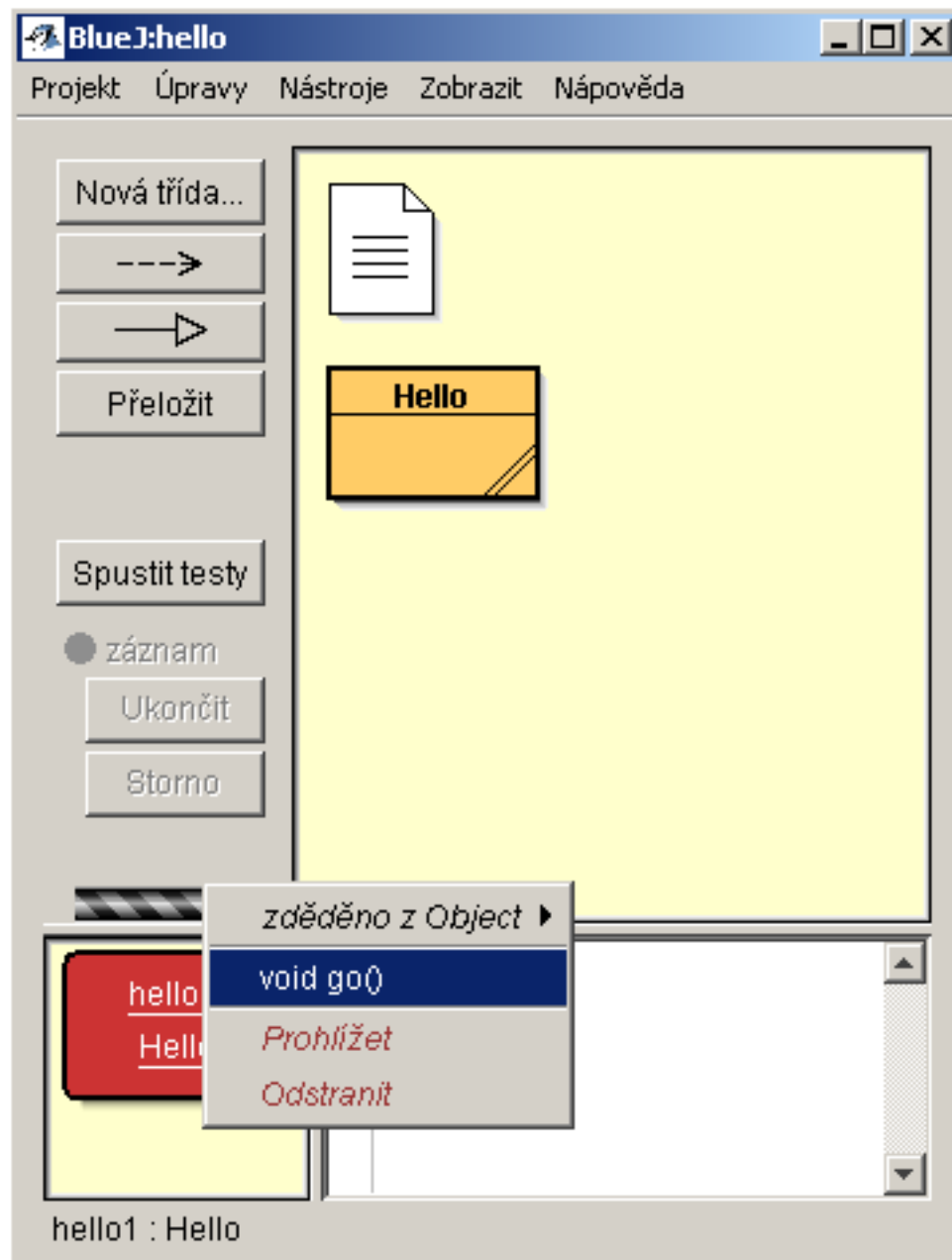
- Vytvořený objekt (instance `hello1` třídy `Hello`) je k dispozici jako červená krabička vlevo dole.

**Obrázek 1.11. Vytvořená instance třídy `Hello`**



- S instancí je nyní možné komunikovat/interagovat - tj. volat metody.
- Jelikož v úvahu (viz zdrojový kód Hello) připadá jen metoda *void go()*, zavoláme právě ji:

**Obrázek 1.12. Volání (spuštění) metody go()**



- Metoda proběhne s výsledkem podobným, jako když jsme program spouštěli metodou main.

**Obrázek 1.13. Výsledek spuštění go()**



Metoda `main` totiž realizovala postupně stejné kroky, jako jsme teď provedli interaktivně (krok po kroku) sami.

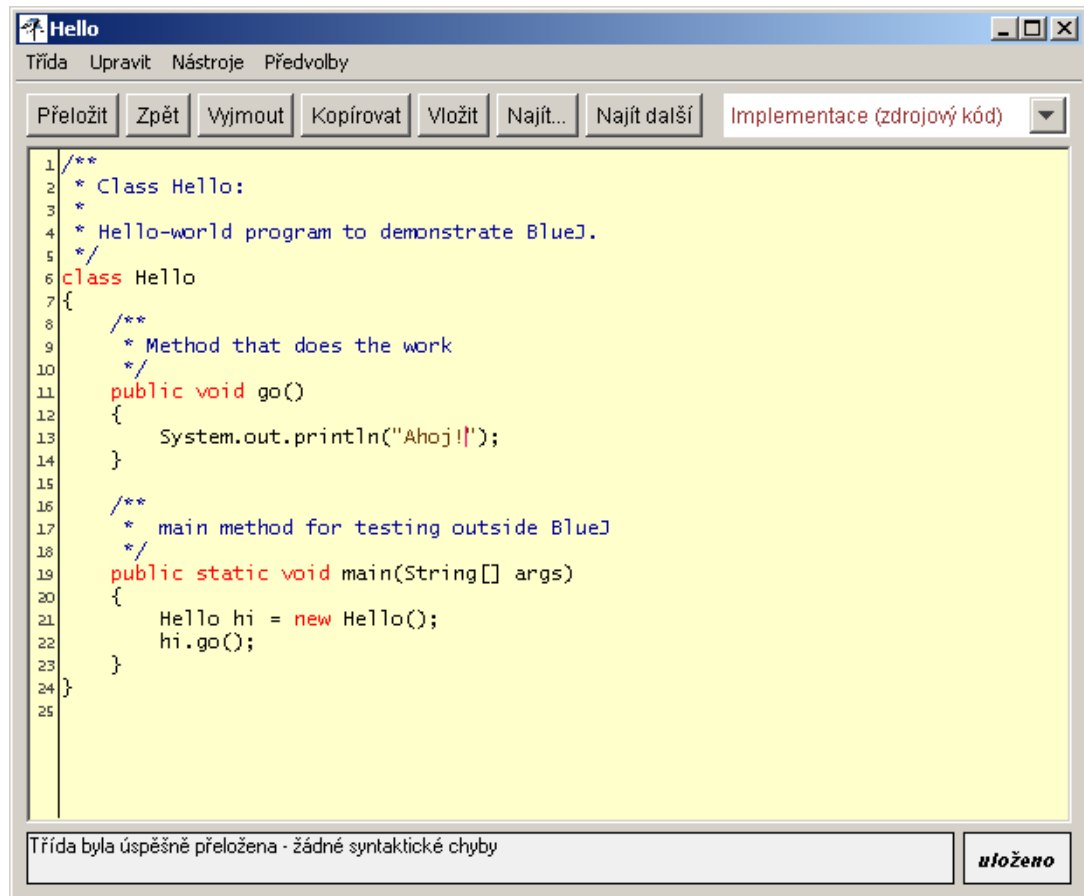
## Úprava hotového programu v BlueJ

Nejjednodušší úpravou programu je změna těla některé z metod, v našem případě již známé metody `go()`.

- Editor zdrojového kódu třídy `Hello` aktivujeme pravým tlačítkem - položkou *Otevřít v editoru*.
- Kód metody `go()` změníme přepsáním hlásky "Hello, world" na "Ahoj!".

Po kliknutí na tlačítko Přeložit by se mělo objevit:

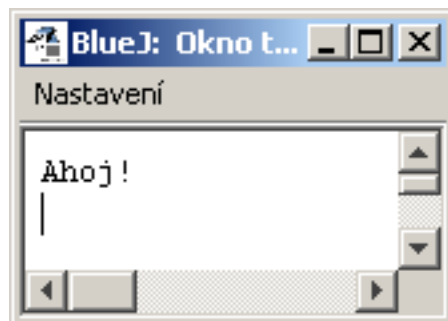
**Obrázek 1.14. Metoda `go()` upravena a třída úspěšně přeložena**



- Třidu Hello s upravenou metodou spustíme jako v předchozím - např. pravým tlačítkem a metodou main.

Objeví se:

**Obrázek 1.15. Výsledek běhu upravené třídy Hello**



## Sestavení a spuštění programu bez BlueJ

- Cílem je porozumět obecné struktuře javového programu
- a být schopni vytvořit a spustit jej i mimo prostředí BlueJ.

## Základní životní cyklus javového programu - bez BlueJ

- Program sestává z jedné (ale obvykle více) **tříd** (class). Ukážeme na příkladu třídy s názvem *NazevTridy*:
- Zdrojový kód každé (veřejně přístupné) třídy je umístěn v jednom souboru (NazevTridy.java  [http://www.instantweb.com/foldoc/foldoc.cgi?NazevTridy.java])
- Postup:
  - vytvoření zdrojového textu (libovolným editorem čistého textu) -> **NazevTridy.java**   
[http://www.instantweb.com/foldoc/foldoc.cgi?NazevTridy.java]
  - překlad (nástrojem javac  [http://www.instantweb.com/foldoc/foldoc.cgi?javac]) **NazevTridy.java**  [http://www.instantweb.com/foldoc/foldoc.cgi?NazevTridy.java] -> **NazevTridy.class**  [http://www.instantweb.com/foldoc/foldoc.cgi?NazevTridy.class]
  - spuštění, např. **java NazevTridy**  [http://www.instantweb.com/foldoc/foldoc.cgi?java NazevTridy]
- překládá se javac NazevTridy.java   
[http://www.instantweb.com/foldoc/foldoc.cgi?javac NazevTridy.java ] (název souboru se třídou včetně přípony .java  [http://www.instantweb.com/foldoc/foldoc.cgi?.java]!!!)
- spouští se vždy udáním java NazevTridy   
[http://www.instantweb.com/foldoc/foldoc.cgi?java NazevTridy] (název třídy bez přípony .class  [http://www.instantweb.com/foldoc/foldoc.cgi?.class]!!!)




## Demo "Ahoj!"

- Půjde o podobný jednoduchý demoprogram, jaký jsme spouštěli v BlueJ pod názvem *Hello*.
- Navíc se zde objeví použití *balíků* - třída bude umístěna do balíku `tomp.ucebnice`.
- Zdrojový kód bude vypadat takto:

```
package tomp.ucebnice;  
public class Pozdrav {
```



```
// Program spouštíme aktivací funkce "main"
public static void main(String[] args) {
    System.out.println("Ahoj!");
}
}
```




Pozn: Jelikož třída Pozdrav  [<http://www.instantweb.com/foldoc/foldoc.cgi?Pozdrav>] je umístěna do `tomp.ucebnice`  [<http://www.instantweb.com/foldoc/foldoc.cgi?tomp.ucebnice>], její zdrojový soubor musí být uložen v podadresáři `tomp\ucebnice`  [<http://www.instantweb.com/foldoc/foldoc.cgi?tomp\ucebnice>].

## Překlad a spuštění "Ahoj!"

### Překlad

1. Máme nainstalován J2SDK 5.0
2. Jsme v adresáři `c:\devel\pb162`  [<http://www.instantweb.com/foldoc/foldoc.cgi?c:\devel\pb162>], v něm je podadresář `tomp\ucebnice`  [<http://www.instantweb.com/foldoc/foldoc.cgi?tomp\ucebnice>], v něm je soubor `Pozdrav.java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Pozdrav.java>]
3. Spustíme *překlad* `javac tomp\ucebnice\Pozdrav.java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?javac tomp\ucebnice\Pozdrav.java>]
4. Je-li program správně napsán, přeloží se "mlčky"
5. (výsledný `.class`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.class>] soubor bude v téže adresáři jako zdroj)

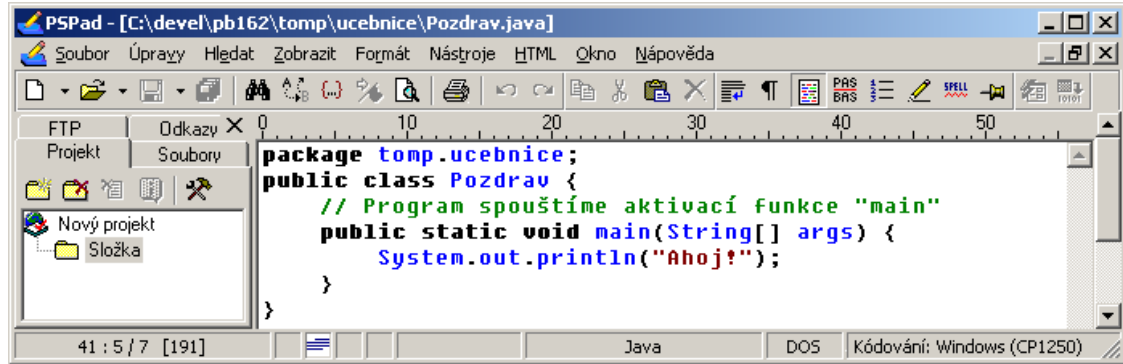
### Spuštění

1. Poté spustíme *program* `Pozdrav: java -classpath . tomp.ucebnice.Pozdrav`  [<http://www.instantweb.com/foldoc/foldoc.cgi?java -classpath . tomp.ucebnice.Pozdrav>]
2. Volba překladače `-classpath` *adresář* zajistí, že (dříve přeložené) třídy používané při spuštění této třídy budou přístupné pod adresářem *adresář*.
3. `-classpath .` tedy značí, že třídy (soubory `.class`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.class>]) se budou hledat v odpovídajících podadresářích aktuálního adresáře (adresáře `.`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.>])


4. Je-li program správně napsán a přeložen, vypíše se Ahoj !

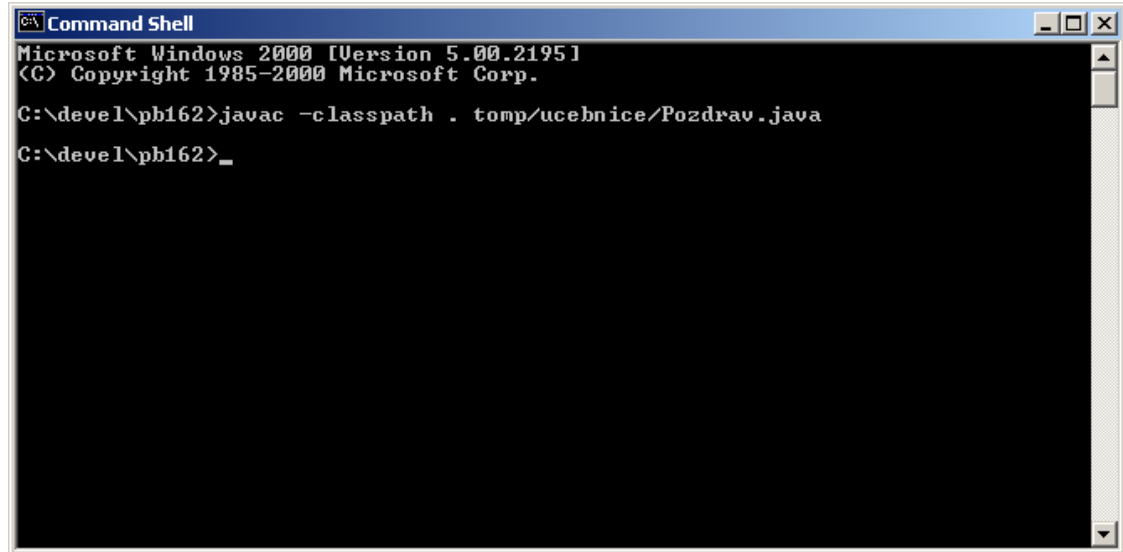
## Vytvoření zdrojového textu "Ahoj!" ("for dummies")

Vytvoření a editace zdrojového kódu v editoru PSPad [<http://pspad.zde.cz>] (dostupný zdarma, instalovatelný na všech Win strojích v učebnách na FI)




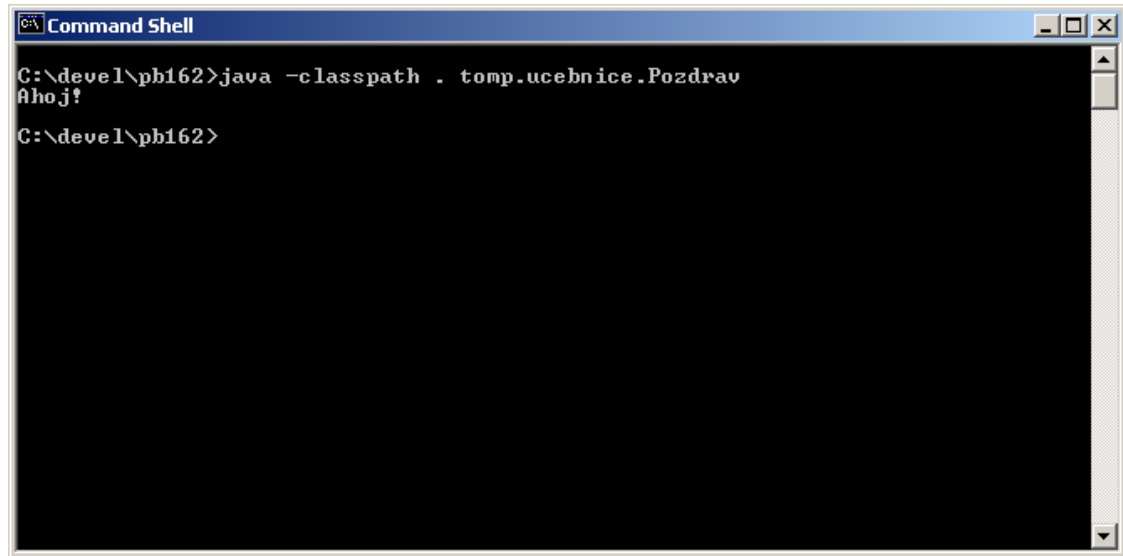
## Překlad "Ahoj!" ("for dummies")

Překlad překladačem **javac**  [<http://www.instantweb.com/foldoc/foldoc.cgi?javac>] (úspěšný, bez hlášení překladače)



## Spuštění "Ahoj!" ("for dummies")

Spuštění voláním **java**  [<http://www.instantweb.com/foldoc/foldoc.cgi?java>]



```

C:\devel\pb162>java -classpath . temp.ucebnice.Pozdrav
Ahoj!
C:\devel\pb162>



```



## Co znamená spustit program?

Spuštění javového programu

= **spuštění metody main jedné ze tříd tvořících program**



Tato funkce může mít parametry:

- podobně jako např. v Pascalu nebo v C
- jsou typu `String`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?String\]](http://www.instantweb.com/foldoc/foldoc.cgi?String) (řetězec)
- předávají se při spuštění z příkazového řádku do pole `String[] args`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?String\[\] args \]](http://www.instantweb.com/foldoc/foldoc.cgi?String[]%20args)

Metoda `main`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?main\]](http://www.instantweb.com/foldoc/foldoc.cgi?main) nevrací žádnou hodnotu - návratový typ je vždy(!) `void`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?void\]](http://www.instantweb.com/foldoc/foldoc.cgi?void)






Její hlavička musí *vypadat vždy přesně tak*, jako ve výše uvedeném příkladu, jinak nebude spuštěna!

## Praktické informace (aneb co je nutné udělat)

Cesty ke spustitelným programům (PATH)  [\[http://www.instantweb.com/foldoc/foldoc.cgi?\(PATH\)\]](http://www.instantweb.com/foldoc/foldoc.cgi?(PATH)) musejí obsahovat i adresář `JAVA_HOME\bin`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?JAVA\\_HOME\bin\]](http://www.instantweb.com/foldoc/foldoc.cgi?JAVA_HOME%5Cbin)

## Praktické informace (aneb co je vhodné udělat)

Systémové proměnné by měly obsahovat:

- JAVA\_HOME=kořenový adresář instalace   
[[http://www.instantweb.com/foldoc/foldoc.cgi?JAVA\\_HOME=kořenový adresář instalace](http://www.instantweb.com/foldoc/foldoc.cgi?JAVA_HOME=kořenový%20adresář%20instalace)] Javy,  
např. JAVA\_HOME=c:\jdk5.0   
[[http://www.instantweb.com/foldoc/foldoc.cgi?JAVA\\_HOME=c:\jdk5.0](http://www.instantweb.com/foldoc/foldoc.cgi?JAVA_HOME=c:\jdk5.0)]
- CLASSPATH=cesty ke třídám   
[[http://www.instantweb.com/foldoc/foldoc.cgi?CLASSPATH=cesty ke třídám](http://www.instantweb.com/foldoc/foldoc.cgi?CLASSPATH=cesty%20ke%20třídám)] (podobně jako v  
PATH  [<http://www.instantweb.com/foldoc/foldoc.cgi?PATH>] jsou cesty ke spustitelným souborům),  
např. CLASSPATH=c:\devel\pb162   
[<http://www.instantweb.com/foldoc/foldoc.cgi?CLASSPATH=c:\devel\pb162>]

## Odkazy

- Odkazy na zdroje (učebnice) <http://www.fi.muni.cz/~tomp/java/ucebnice/resources.html>
- Další tutoriály: <http://www.mike-levin.com/learning-java/toc.html>

---

# Kapitola 2. Základní pojmy OOP. Třída, objekt, jeho vlastnosti. Metody, proměnné. Konstruktory.

## Úvod do objektového programování

- Pojmy: třída, objekt
- Deklarace a definice tříd, jejich vlastnosti (proměnné, metody)
- Vytváření objektů (deklarace sama objekt nevytvoří...), proměnné odkazující na objekt
- Jmenné konvence - jak tvořit jména tříd, proměnných, metod
- Použití objektů, volání metod, přístupy k proměnným
- Modifikátory přístupu/viditelnosti (public, protected...)
- Konstruktory (dotvoří/naplní prázdný objekt)
- Přetěžování metod (dvě metody se stejným názvem a jinými parametry)

## Třída, objekt, jejich vlastnosti

### Co je třída a objekt?

**Třída** (také poněkud nepřesně zvaná *objektový typ*) představuje skupinu objektů, které nesou stejné vlastnosti







"stejně" je myšleno *kvalitativně*, nikoli *kvantitativně*, tj.

- např. všechny objekty třídy `Clovek`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>] mají vlastnost `jmeno`  [<http://www.instantweb.com/foldoc/foldoc.cgi?jmeno>],
- tato vlastnost má však obecně pro různé lidi různé hodnoty - lidi mají různá jména

**Objekt** je jeden konkrétní jedinec (reprezentant, entita) příslušné třídy

pro konkrétní objekt **nabývají vlastnosti** deklarované třídou **konkrétních hodnot**

Příklad:

- Třída `Clovek`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>] má vlastnost `jmeno`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?jmeno>]
- Objekt `panProfesor`  [<http://www.instantweb.com/foldoc/foldoc.cgi?panProfesor>] typu `Clovek`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>] má vlastnost `jmeno`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?jmeno>] s hodnotou "Václav Klaus"   
[[http://www.instantweb.com/foldoc/foldoc.cgi?\"Václav Klaus\"](http://www.instantweb.com/foldoc/foldoc.cgi?\)].

## Vlastnosti objektu (1)

Vlastnostmi objektů jsou:

- *proměnné*
- *metody*

Vlastnosti objektů - proměnné i metody - je třeba *deklarovat*.

viz Sun Java Tutorial / Trail: Learning the Java Language: Lesson: Classes and Inheritance  
[<http://java.sun.com/docs/books/tutorial/java/javaOO/classes.html>]

## Vlastnosti objektu (2)

Proměnné

1. jsou nositeli "pasivních" vlastností; jakýchsi *atributů*, *charakteristik* objektů
2. de facto jde o datové hodnoty svázané (zapouzdřené) v objektu

Metody

1. jsou nositeli "výkonných" vlastností; "dovedností" objektů
2. de facto jde o funkce (procedury) pracující (převážně) nad proměnnými objektu

## Deklarace a použití třídy

### Příklad - deklarace třídy `Clovek`


- deklarujeme třídu objektů - lidí
- ```
public class Clovek {
```

```
protected String jmeno;
protected int rokNarozeni;

public Clovek(String j, int rN) {
    jmeno = j;
    rokNarozeni = rN;
}

public void vypisInfo() {
    System.out.println("Clovek:");
    System.out.println("Jmeno="+jmeno);
    System.out.println("Rok narozeni="+rokNarozeni);
}
}
```

- Použijme ji v programu -


1. vytvoříme instanci - objekt - typu Clovek   
[<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>]
2. vypíšeme informace o něm

## Příklad - použití třídy Clovek (1)

Mějme deklarovanou třídu *Clovek*

Metoda *main* v následujícím programu:

1. vytvoří 2 lidi (pomocí *new Clovek*)
2. zavolá jejich metody

`vypisInfo()`  [[http://www.instantweb.com/foldoc/foldoc.cgi?vypisInfo\(\)](http://www.instantweb.com/foldoc/foldoc.cgi?vypisInfo())]

```
public class TestLidi {
    public static void main(String[] args) {
        Clovek ales = new Clovek("Ales Necas", 1966);
        Clovek beata = new Clovek("Beata Novakova", 1970);
        ales.vypisInfo();
        beata.vypisInfo();
    }
}
```


Tedy: vypíší se informace o obou vytvořených objektech - lidech.


Nyní podrobněji k *proměnným* objektů.

## Příklad - použití třídy Clovek (2)

Ve výše uvedeném programu znamenalo na řádku:

```
Clovek ales = new Clovek("Ales Necas", 1966);
```


**Clovek ales**  [http://www.instantweb.com/foldoc/foldoc.cgi?Clovek ales]: pouze deklarace (tj. určení typu) proměnné *ales* - bude typu *Clovek*.

**ales = new Clovek ("Ales Necas", 1966)**  [http://www.instantweb.com/foldoc/foldoc.cgi?ales = new Clovek ("Ales Necas", 1966)]: vytvoření objektu *Clovek* se jménem *Ales Necas*.

Lze napsat zvlášť do dvou řádků nebo (tak jak jsme to udělali) na řádek jeden.

Každý příkaz i deklaraci ukončujeme středníkem.


## Vytváření objektů

Ve výše uvedených příkladech jsme objekty vytvářeli voláními **new Clovek(...)**  [http://www.instantweb.com/foldoc/foldoc.cgi?new Clovek(...)] bezděčně jsme tak použili

- **operátor new**, který vytvoří prázdný objekt typu *Clovek* a
- volání **konstruktoru**, který prázdný objekt *naplní počátečními údaji* (daty).

## Shrnutí




Objekty:

- jsou instance "své" třídy
- vytváříme je operátorem **new**  [http://www.instantweb.com/foldoc/foldoc.cgi?new] - voláním konstrukturu
- vytvořené objekty ukládáme do proměnné stejného typu (nebo typu předka či implementovaného rozhraní - o tom až později)

## Proměnné


### Proměnné - deklarace




Položky `jmeno`  [<http://www.instantweb.com/foldoc/foldoc.cgi?jmeno>] a `rokNarozeni`  [<http://www.instantweb.com/foldoc/foldoc.cgi?rokNarozeni>] v předchozím příkladu jsou **proměnné** objektu `Clovek`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>].

Jsou deklarovány v těle deklarace třídy `Clovek`.


Deklarace proměnné objektu má tvar:

```
modifikátory Typ jméno;  [http://www.instantweb.com/foldoc/foldoc.cgi?modifikátory Typ jméno;]
```

např.:

```
protected int rokNarozeni;  [http://www.instantweb.com/foldoc/foldoc.cgi?protected int rokNarozeni;]
```

## Proměnné - datový typ

Výše uvedená proměnná `rokNarozeni`  [<http://www.instantweb.com/foldoc/foldoc.cgi?rokNarozeni>] měla datový typ `int` (32bitové celé číslo). Tedy:

- proměnná takového typu nese *jednu hodnotu typu celé číslo* (v rozsahu  $-2^{31}.. 2^{31}-1$ );
- nese-li jednu hodnotu, pak se jedná o tzv. **primitivní datový typ**.

Kromě celých čísel `int` nabízí Java celou řadu dalších primitivních datových typů. Primitivní typy jsou dané napevno, programátor je jen používá, nedefinuje.

Tam, kde nestačí diskrétní hodnoty (tj. primitivní typy), musíme použít typy *složené*, **objektové**.

- Objektovými typy v Javě jsou **třídy** (class) a **rozhraní** (interface). Třídy už jsme viděli v příkladu `Clovek`.
- Existují třídy definované přímo v Javě, v knihovně Java Core API.
- Nenajdeme-li tam třídu, kterou potřebujeme, můžeme si ji nadefinovat sami - viz `Clovek`.


## Proměnné - jmenné konvence

Na jméno (identifikátor) proměnné sice Java neklade žádná speciální omezení (tedy mimo omezení platná pro jakýkoli identifikátor), ale přesto bývá velmi dobrým zvykem dodržovat při pojmenovávání následující pravidla (blíže viz podrobný rozbor na FIXME):


- jména začínají malým písmenem

- nepoužíváme diakritiku (problémy s editory, přenositelností a kódováním znaků)
- (raději ani český jazyk, angličtině rozumí "každý")
- je-li to složenina více slov, pak je *nespojujeme podtržítkem*, ale další začne velkým písmenem (tzv. "CamelCase")

např.:

```
protected int rokNarozeni;  [http://www.instantweb.com/foldoc/foldoc.cgi? protected int  
rokNarozeni;]
```

je identifikátor se správně (vhodně) utvořeným jménem, zatímco:

```
protected int RokNarozeni;  [http://www.instantweb.com/foldoc/foldoc.cgi? protected int  
RokNarozeni;]
```

není vhodný identifikátor proměnné (začíná velkým písmenem)

**Dodržování těchto jmenných konvencí je základem psaní srozumitelných programů a bude vyžadováno, sledováno a hodnoceno v odevzdávaných úlohách i písémkách.**

## Proměnné - použití

Proměnné objektu odkazujeme pomocí "tečkové notace":

```
public class TestLidi2 {  
    public static void main(String[] args) {  
        ...  
        Clovek ales = new Clovek("Ales Necas", 1966); // vytvoření objektu ...  
        System.out.println(ales.jmeno); // přístup k (čtení) jeho proměnné ...  
        ales.jmeno = "Aleš Novák"; // modifikace (zápis do) jeho proměnné  
    }  
}
```

## Proměnné - modifikátory přístupu

Přístup k proměnným (i metodám) může být řízen uvedením tzv. *modifikátorů* před deklaraci prvku, viz výše:

```
// protected = přístup pouze z třídy ve stejném balíku nebo z podtřídy:  
protected String jmeno;
```

*Modifikátorů* je více typů, nejběžnější jsou právě zmíněné *modifikátory přístupu* (přístupových práv)

## Proměnné - použití (2)

Objektů (tzv. *instancí*) stejného typu (tj. stejné třídy) si můžeme postupně vytvořit více:

```
public class TestLidi3 {  
    public static void main(String[] args) {  
        ...  
        Clovek ales = new Clovek("Ales Necas", 1966); // vytvoření prvního objektu  
        Clovek petr = new Clovek("Petr Svoboda", 1968); // vytvoření druhého objektu  
        System.out.println(ales.jmeno); // přístup k (čtení) proměnné - prvnímu objektu  
        System.out.println(petr.jmeno); // přístup k (čtení) proměnné - druhému objektu  
    }  
}
```

Existují tedy dva objekty, každý má své (obecně různé) hodnoty proměnných - např. jsou různá jména obou lidí.

## Metody

### Metody

Nad *existujícími* (vytvořenými) objekty můžeme volat jejich *metody*. Metoda je:

- podprogram (funkce, procedura), který *primárně pracuje s proměnnými "mateřského" objektu*,
- může mít další *parametry*
- může *vracet hodnotu* podobně jako v Pascalu *funkce*.

Každá metoda se musí ve své třídě *deklarovat*.

V Javě *neexistují metody deklarované mimo třídy* (tj. Java nezná žádné "globální" metody).

## Metody - příklad

Výše uvedená třída Clovek měla metodu na výpis informací o daném objektu/člověku:

```
public class Clovek {  
    protected String jmeno;  
    protected int rokNarozeni;  
  
    public Clovek(String j, int rN) {  
        jmeno = j;  
        rokNarozeni = rN;  
    }  
    // Metoda vypisInfo() na výpis informací o člověku:  
    public void vypisInfo() {  
        System.out.println("Clovek:");  
    }  
}
```

```
        System.out.println("Jmeno="+jmeno);  
        System.out.println("Rok narozeni="+rokNarozeni);  
    }  
}
```

## Volání metod

- *Samotnou deklarací* (napsáním kódu) metody *se žádný kód neprovede*.
- Chceme-li vykonat kód metody, musíme ji *zavolat*.
- Volání se realizuje (tak jako u proměnných) "*tečkovou notací*", viz dále.
- Volání lze provést, jen je-li metoda z místa volání přístupná - "viditelná". Přístupnost regulují podobně jako u proměnných modifikátory přístupu.

## Volání metod - příklad

Vracíme se k prvnímu příkladu: vytvoříme dva lidi a zavoláme postupně jejich metodu

*vypisInfo*

.

```
public class TestLidi {  
    public static void main(String[] args) {  
        Clovek ales = new Clovek("Ales Necas", 1966);  
        Clovek beata = new Clovek("Beata Novakova", 1970);  
        ales.vypisInfo(); // volání metody objektu ales  
        beata.vypisInfo(); // volání metody objektu beata  
    }  
}
```

Vytvoří se dva objekty Clovek a vypíší se informace o nich.

## Parametry metod

V deklaraci metody uvádíme v její hlavičce tzv. *formální parametry*.

Syntaxe:


```
modifikatory typVraceneHodnoty nazevMetody(seznamFormalnichParametru) {  
    tělo (výkonný kód) metody  
}
```

*seznamFormalnichParametru*

je

tvaru:



[<http://www.instantweb.com/foldoc/foldoc.cgi?seznamFormalnichParametru> je tvaru: ] typParametru  
navezFormalnihoParametru, ...   
[<http://www.instantweb.com/foldoc/foldoc.cgi?typParametru> navezFormalnihoParametru, ...]

Podobně jako v jiných jazycích parametr představuje v rámci metody *lokální proměnnou*.

Při volání metody jsou f. p. nahrazeny *skutečnými parametry*.

## Předávání skutečných parametrů metodám

Hodnoty primitivních typů - čísla, logické hodnoty, znaky

- se předávají **hodnotou**, tj. hodnota se nakopíruje do lokální proměnné metody

Hodnoty objektových typů - všechny ostatní (tj. vč. všech uživatelem definovaných typů)

- se předávají **odkazem**, tj. do lokální proměnné metody se nakopíruje **odkaz na objekt - skutečný parametr**

Pozn: ve skutečnosti se tedy parametry *vždy předávají hodnotou*, protože v případě objektových parametrů se předává *hodnota odkazu na objekt - skutečný parametr*.

V Javě tedy nemáme jako programátoři moc na výběr, jak parametry předávat

- to je ale spíše výhoda!

## Příklad předávání parametrů - primitivní typy

Rozšířme definici třídy

*Clovek*

o metodu

*zakric*

s parametry:

```
...
public void zakric(int kolikrat) {
    System.out.println("Kricim " + kolikrat + "krat UAAAA!");
}
...
```

Při zavolání:

```
...  
zakric(10);  
...
```

tato metoda vypíše

Kricim 10krat UAAAA!

## Příklad předávání parametrů - objektové typy (1)

Následující třída *Ucet* modeluje jednoduchý bankovní účet s možnostmi:

- přidávat na účet/odebírat z účtu
- vypisovat zůstatek na něm
- převádět na jiný účet

```
public class Ucet {  
    // stav (zustatek) penez uctu  
    protected double zustatek;  
  
    public void pridej(double castka) {  
        zustatek += castka;  
    }  
  
    public void vypisZustatek() {  
        System.out.println(zustatek);  
    }  
  
    public void prevedNa(Ucet kam, double castka) {  
        zustatek -= castka;  
        kam.pridej(castka);  
    }  
}
```

Metoda *prevedNa* pracovat nejen se svým "mateřským" objektem, ale i s objektem *kam* předaným do metody... opět přes tečkovou notaci.

## Příklad předávání parametrů - objektové typy (2)


Příklad použití třídy


*Ucet*

:



```
...
public static void main(String[] args) {
    Ucet petruvUcet = new Ucet();
    Ucet ivanuvUcet = new Ucet();
    petruvUcet.pridej(100);
    ivanuvUcet.pridej(220);
    petruvUcet.prevedNa(ivanuvUcet, 50);
    petruvUcet.vypisZustatek();
    ivanuvUcet.vypisZustatek();
}
```

## Návrat z metody

Kód metody skončí, tj. předá řízení zpět volající metodě (nebo systému - v případě startovní metody `main`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?main\]](http://www.instantweb.com/foldoc/foldoc.cgi?main)), jakmile

- dokončí poslední příkaz v těle metody nebo
- dospěje k příkazu `return`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?return\]](http://www.instantweb.com/foldoc/foldoc.cgi?return)

Metoda může při návratu *vrátit hodnotu* - tj. chovat se jako *funkce* (ve pascalském smyslu):


- Vrácenou hodnotu musíme uvést za příkazem `return`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?return\]](http://www.instantweb.com/foldoc/foldoc.cgi?return). V tomto případě tedy nesmí `return` chybět!
- Typ vrácené hodnoty musíme v *hlavičce metody deklarovat*.
- Nevrací-li metoda nic, pak musíme namísto typu vrácené hodnoty psát `void`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?void\]](http://www.instantweb.com/foldoc/foldoc.cgi?void).

Pozn.: I když metoda něco vrátí, my to nemusíme použít, ale je to trochu divné...

## Konstruktory

### Konstruktory

Co a k čemu jsou konstruktory?

- Konstruktory jsou speciální *metody* volané při *vytváření nových instancí* dané třídy.
- Typicky se v konstruktoru *naplní (inicializují) proměnné objektu*.
- Konstruktory lze volat jen ve spojení s operátorem `new` 

[<http://www.instantweb.com/foldoc/foldoc.cgi?new>] k vytvoření nové instance třídy - nového objektu, evt. volat z jiného konstruktoru

Syntaxe (viz výše):

```
public class Clovek {
    protected String jmeno;
    protected int rokNarozeni;
    // konstruktor se dvěma parametry
    // - inicializuje hodnoty proměnných ve vytvořeném objektu
    public Clovek(String j, int rN) {
        jmeno = j;
        rokNarozeni = rN;
    }
    ...
}
```


Příklad využití tohoto konstruktoru:

```
...
Clovek pepa = new Clovek("Pepa z Hongkongu", 1899);
...
```

Toto volání vytvoří objekt pepa a naplní ho jménem a rokem narození.

## Konstruktory (2)

Jak je psát a co s nimi lze dělat?

- nemají návratový typ (ani void  [<http://www.instantweb.com/foldoc/foldoc.cgi?void>] - *to už vůbec ne!!!*)
- mohou mít parametry
- mohou volat konstruktor rodičovské třídy - ale jen jako svůj první příkaz

## Přetěžování metod

### Přetěžování



Jedna třída může mít:

- Více metod se *stejnými názvy, ale různými parametry*.



- Pak hovoříme o tzv. *přetížené* (overloaded) metodě.
- Nelze přetížit metodu *pouze změnou typu návratové hodnoty*.

## Přetěžování - příklad

Ve třídě  [http://www.instantweb.com/foldoc/foldoc.cgi?Ucet] přetížíme metodu prevedNa  [http://www.instantweb.com/foldoc/foldoc.cgi?prevedNa].

- Přetížená metoda převede na účet příjemce celý zůstatek z účtu odesílatele:

```
public void prevedNa(Ucet u) {  
    u.pridej(zustatek);  
    zustatek = 0;  
}
```

Ve třídě

*Ucet*

koexistují dvě různé metody se stejným názvem, ale jinými parametry.

Pozn: I když jsou to teoreticky dvě úplně různé metody, pak *když už se jmenují stejně, měly by dělat něco podobného*.

## Přetěžování - příklad (2)

- Často přetížená metoda volá jinou "verzi" metody se stejným názvem:

```
public void prevedNa(Ucet u) {  
    prevedNa(u, zustatek);  
}
```

- Toto je *jednodušší, přehlednější*, udělá se tam potenciálně méně chyb.

Lze doporučit. Je to přesně postup divide-et-impera, rozděl a panuj, dělba práce mezi metodami!

## Přetěžování - příklad (3)

- Je ale otázka, zdali převod celého zůstatku raději nenapsat jako *nepřetíženou*, samostatnou metodu, např.:

```
public void prevedVseNa(Ucet u) {
```

```
        prevedNa(u, zustatek);  
    }
```


- Je to o něco instruktivnější, ale přibude další identifikátor - název metody - k zapamatování.

Což může být výhoda (je to výstižné) i nevýhoda (musíme si pamatovat další).

## Odkazy na objekty (instance)

### Odkazy na objekty (instance)


Deklarace proměnné objektového typu ještě žádný objekt nevytváří.


To se děje až příkazem - operátorem - **new**  [<http://www.instantweb.com/foldoc/foldoc.cgi?new>].

- Proměnné objektového typu jsou vlastně **odkazy na dynamicky vytvořené objekty**.
- Přiřazením takové proměnné zkopírujeme pouze odkaz, nikoli celý objekt.

## Přiřazování objektových proměnných

V následující ukázce vytvoříme dva účty.

- Odkazy na ně budou primárně v proměnných *petruvUcet* a *ivanuvUcet*.
- V proměnné *u* nebude primárně odkaz na žádný účet.
- Pak do ní přiřadíme (***u = petruvUcet;***  [[http://www.instantweb.com/foldoc/foldoc.cgi?u=petruvUcet](http://www.instantweb.com/foldoc/foldoc.cgi?u=petruvUcet;)];) odkaz na objekt skrývající se pod odkazem *petruvUcet*.
- Od této chvíle můžeme s účtem *petruvUcet* manipulovat přes odkaz (proměnnou) *u*.


Což se také děje: ***u.prevedNa(ivanuvUcet, 50);***   
[[http://www.instantweb.com/foldoc/foldoc.cgi?u.prevedNa\(ivanuvUcet, 50\);](http://www.instantweb.com/foldoc/foldoc.cgi?u.prevedNa(ivanuvUcet, 50);)]



```
...  
public static void main(String[] args) {  
    Ucet petruvUcet = new Ucet();  
    Ucet ivanuvUcet = new Ucet();  
    Ucet u;  
    petruvUcet.pridej(100);  
    ivanuvUcet.pridej(220);  
    u = petruvUcet;  
}
```

```
u.prevedNa(ivanuvUcet, 50); // odečte se z Petrova účtu
petruvUcet.vypisZustatek(); // vypíše 50
ivanuvUcet.vypisZustatek();
}
```

## Vracení odkazu na sebe

Metoda může vracet odkaz na objekt, nad nímž je volána pomocí

```
return this;  [http://www.instantweb.com/foldoc/foldoc.cgi? return this; ]
```

Příklad - upravený Ucet  [http://www.instantweb.com/foldoc/foldoc.cgi?Ucet] s metodou preved-  
Na  [http://www.instantweb.com/foldoc/foldoc.cgi?prevedNa] vracející odkaz na sebe

```
public class Ucet {
    float zustatek;

    public void pridej(float castka) {
        zustatek += castka;
    }

    public void vypisZustatek() {
        System.out.println(zustatek);
    }

    public Ucet prevedNa(Ucet u, float castka) {
        zustatek -= castka; // nebo také vhodné je: pridej(-castka);
        u.pridej(castka);
        return this;
    }
}
```

## Řetězení volání

Vracení odkazu na sebe (tj. na objekt, na němž se metoda volala) lze s výhodou využít k "řetězení" volání:


```
...
public static void main(String[] args) {
    Ucet petruvUcet = new Ucet();
    Ucet ivanuvUcet = new Ucet();
    Ucet igoruvUcet = new Ucet();
    petruvUcet.pridej(100);
    ivanuvUcet.pridej(100);
    igoruvUcet.pridej(100);
    // budeme řetězit volání:
}
```

```
petruvUcet.prevedNa(ivanuvUcet, 50).prevedNa(igoruvUcet, 20);  
petruvUcet.vypisZustatek(); // vypíše 30  
ivanuvUcet.vypisZustatek(); // vypíše 150  
igoruvUcet.vypisZustatek(); // vypíše 120  
}
```

## Statické proměnné a metody

### Proměnné a metody třídy - statické

Dosud jsme zmiňovali **proměnné a metody** (tj. souhrnně *prvky - members*) **objektu**.

Lze deklarovat také metody a proměnné patřící *celé třídě*, tj. skupině všech objektů daného typu. Takové metody a proměnné nazýváme **statické** a označujeme v deklaraci modifikátorem `static` 

[<http://www.instantweb.com/foldoc/foldoc.cgi?static>]

### Příklad statické proměnné a metody

Představme si, že si budeme pamatovat, kolik lidí se nám během chodu programu vytvořilo a vypisovat tento počet.

Budeme tedy potřebovat do třídy *Clovek* doplnit:

- jednu proměnnou *pocetLidi* společnou pro celou třídu *Clovek* - každý člověk ji při svém vzniku zvýší o jedna.
- jednu metodu *kolikMamLidi*, která vrátí počet dosud vytvořených lidí.

```
public class Clovek {  
  
    protected String jmeno;  
    protected int rokNarozeni;  
    protected static int pocetLidi = 0;  
  
    public Clovek(String j, int rN) {  
        jmeno = j;  
        rokNarozeni = rN;  
        pocetLidi++;  
    }  
    ...  
    public static intolikMamLidi() {  
        return pocetLidi;  
    }  
    ...  
}
```

}

Pozn: Všimněte si v obou případech modifikátoru/klíčového slova *static*.

---

# Kapitola 3. Struktura složitějších programů. Rozhraní. Dědičnost.

## Objektové modelování reality

- Kroky řešení problému na počítači - pár slov o SW inženýrství

### Kroky řešení reálného problému na počítači

Generický (univerzální, obecný...) model postupu:

1. Zadání problému
2. Shromáždění informací o realitě a jejich analýza
3. Modelování reality na počítači a implementace požadovaných operací nad modelovanou realitou

### Vývoj software je proces...

(podle JS, SW inženýrství):

1. při němž jsou **uživatelské potřeby**
2. transformovány na **požadavky na software**,
3. tyto jsou transformovány na **návrh**,
4. návrh je implementován pomocí **kódu**,
5. kód je **testován**, **dokumentován** a **certifikován** pro operační použití.

### Celkový rámec vývoje SW

V tomto předmětu nás z toho bude zajímat jen něco a jen částečně:

1. **Specifikace** (tj. zadání a jeho formalizace)
2. **Vývoj** (tj. návrh a vlastní programování)
3. částečně **Validace** (z ní především testování)

- 1. **Specifikace SW:** Je třeba definovat funkcionalitu SW a operační omezení.
  2. **Vývoj SW:** Je třeba vytvořit SW, který splňuje požadavky kladené ve specifikaci.
  3. **Validace SW:** SW musí být validován („kolaudován“), aby bylo potvrzeno, že řeší právě to, co požaduje uživatel.
  4. **Evoluce SW:** SW musí být dále rozvíjen, aby vyhověl měnícím se požadavkům zákazníka.

## Metodiky vývoje SW

Tyto generické modely jsou dále rozpracovávány do podoby konkrétních *metodik*.

Metodika (tvorby SW) je ucelený soubor inženýrských postupů, jak řízeným způsobem, s odhadnutelnou spotřebou zdrojů dospět k použitelnému SW produktu.

Některé skupiny metodik:

- strukturovaná
- objektová
- ...

## Metodika typu "vodopád"

Nevracím se nikdy o více jak jednu úroveň zpět:

1. Analýza (Analysis)
2. Návrh (Design)
3. Implementace (Implementation)
4. Testování (Testing)
5. Nasazení (Deployment)

Nyní zpět k Javě a jednoduchým programům...

## Srovnání Java - Pascal

Co bude odlišné oproti dosavadním programátorským zkušenostem?

Struktura a rozsah programu:

|        |                                                                                                                                                                                 |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pascal | program měl jeden nebo více zdrojových souborů (soubor = modul) tvořenými jednotlivými procedurami/fcemi, definicemi a deklaracemi (typů, proměnných...)                        |
| Java   | (a některé další OO jazyky): program je obvykle tvořen více soubory (soubor = popis jedné třídy) tvořenými deklaracemi metod a proměnných (případně dalších prvků) těchto tříd. |

## Organizace programových souborů

- v *Pascalu*: zdrojové (`.pas`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.pas>]) soubory, výsledný (jeden) spustitelný soubor (`.exe`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.exe>]), resp. přeložené kódy jednotek (`.tpu`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.tpu>])
- v *Javě*: zdrojové (`.java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.java>]) soubory, přeložené soubory v bajtkódu (`.class`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.class>]) - jeden z nich spouštíme

## Organizace zdrojových souborů

v *Pascalu* nebyla (nutná)

v *Javě* je nezbytná - zdrojové soubory organizujeme podle toho, ve kterých balících jsou třídy zařazeny (přeložené soubory se *implicitně* ukládají vedle zdrojových)

## Příklad - svět chovatelů a jejich psů

Zkusme naznačit, jak bychom realizovali jednoduchý systém, který bude

1. shromažďovat, ukládat a na požádání zpřístupňovat informace o psech (+ jejich výcviku, očkování...)
2. o jejich chovatelích
3. a dalších souvisejících entitách (např. chovatelských sdruženích, veterinářích,...)

## Shromáždění informací o realitě

Zjistíme, jaké *typy objektů* se ve zkoumaném výseku reality vyskytují a které potřebujeme

- *člověk, pes, veterinář*

Zjistíme a zachytíme vztahy mezi objekty našeho zájmu



- *člověk-chovatel vlastní psa*

Zjistíme, které činnosti objekty (aktéři, aktoři) provádějí

- *veterinář psa očkuje, pes štěká, kousne člověka...*

## Jak zachytíme tyto informace

Jak zachytíme tyto informace:



- neformálními prostředky - tužkou na papíře vlastními slovy v přirozeném jazyce
- formálně pomocí nějakého vyjadřovacího aparátu - např. grafického jazyka
- pomocí CASE nástroje přímo na počítači

Zatím se přidržíme neformálního způsobu...

## Modelování reality pomocí tříd

Určení základních **tříd**, tj.

skupin (kategorií) objektů, které mají podobné vlastnosti/schopnosti:

- Pes  [<http://www.instantweb.com/foldoc/foldoc.cgi?Pes>]
- Clovek  [<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>]
- ...

## Rozhraní

### Rozhraní

V Javě, na rozdíl od C++ neexistuje vícenásobná dědičnost -

- to nám ušetří řadu komplikací
- ale je třeba to něčím nahradit

Pokud po třídě chceme, aby disponovala vlastnostmi z několika různých množin (skupin), můžeme ji de-


klarovat tak, že

- implementuje více rozhraní

## Co je rozhraní

- Rozhraní je vlastně *popis (specifikace) množiny vlastností, aniž bychom tyto vlastnosti ihned implementovali*. Vlastnostmi zde rozumíme především *metody*.
- Říkáme, že určitá třída *implementuje rozhraní*, pokud implementuje (tedy *má* - přímo sama nebo po dědi) všechny vlastnosti (tj. metody), které jsou daným rozhraním předepsány.
- Javové rozhraní je tedy *množina hlaviček metod označená identifikátorem* - názvem rozhraní. (a celých specifikací - tj. popisem, co přesně má metoda dělat - vstupy/výstupy metody, její vedlejší efekty...)

## Deklarace rozhraní

- Vypadá i umísťuje se do souborů podobně jako deklarace třídy
- Všechny metody v rozhraní musí být `public`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?public>] a v hlavičce se to ani nemusí uvádět.
- Těla metod v deklaraci rozhraní se nepiší. (Metody v rozhraní tudíž vypadají velmi podobně jako abstraktní metody ve třídách, ale nemusím psát `abstract`.)


Příklad deklarace rozhraní


```
public interface Informujici {  
    void vypisInfo();  
}
```

## Implementace rozhraní

Příklad

```
public class Clovek implements Informujici {  
    ...  
    public void vypisInfo() {  
        ...  
    }  
}
```

Čteme: Třída Clovek  [http://www.instantweb.com/foldoc/foldoc.cgi?Clovek ]implementuje rozhraní Informujici  [http://www.instantweb.com/foldoc/foldoc.cgi?Informujici].

1. Třída v hlavičce uvede implements NázevRozhraní   
[http://www.instantweb.com/foldoc/foldoc.cgi?implements NázevRozhraní ]
2. Třída implementuje všechny metody předepsané rozhraním

## Využití rozhraní

1. Potřebujeme-li u jisté proměnné právě jen funkcionalitu popsanou určitým rozhraním,
2. tuto proměnnou můžeme pak deklarovat jako typu rozhraní - ne přímo objektu, který rozhraní implementuje.





Příklad

```
Informujici petr = new Clovek("Petr Novák", 1945);  
petr.vypisInfo(); // "petr" stačí deklarovat jen jako Informujici  
                // jiné metody než předepsané tímto intf.  
                // nepotřebujeme!
```

## Dědičnost

### Dědičnost

V realitě jsme často svědci toho, že třídy jsou **podtřídami** jiných:

- tj. všechny objekty podtřídy jsou zároveň objekty nadtřídy, např. každý objekt typu (třídy) ChovatelPsu  [http://www.instantweb.com/foldoc/foldoc.cgi?ChovatelPsu] je současně typu Clovek  [http://www.instantweb.com/foldoc/foldoc.cgi?Clovek] nebo
- např. každý objekt typu (třídy) Pes  [http://www.instantweb.com/foldoc/foldoc.cgi?Pes] je současně typu DomaciZvire  [http://www.instantweb.com/foldoc/foldoc.cgi?DomaciZvire] (alespoň v našem výseku reality - existují i psi "nedomáci"...)

Podtřída je tedy "zjemněním" nadtřídy:

- přebírá její vlastnosti a zpravidla přidává další, **rozšiřuje** svou nadtřidu/předka




V Javě je *každá* uživatelem definovaná třída potomkem nějaké jiné - neuvedeme-li předka explicitně, je předkem vestavěná třída `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]

## Terminologie dědičnosti

Terminologie:


- Nadtřídě (superclass) se také říká "(bezprostřední) předek", "rodičovská třída"
- Podtřídě (subclass) se také říká "(bezprostřední) potomek", "dceřinná třída"

Dědění může mít i více "generací", např.

`Zivocich`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Zivocich>] <- `Clovek`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?Clovek>] <- `Chovatel`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?Chovatel>] (živoch je rodičovskou třídou člověka, ten je rodičovskou třídou chovatele)

Přeneseně tedy předkem (nikoli bezprostředním) chovatele je živoch.

## Jak zapisujeme dědění

Klíčovým slovem `extends`  [<http://www.instantweb.com/foldoc/foldoc.cgi?extends>]:

```
public class Clovek extends Zivocich {  
    ... popis vlastností (proměnných, metod...) člověka navíc oproti živočichovi ..  
}
```

## Dědičnost a vlastnosti tříd

Jak víme, třídy popisují skupiny objektů podobných vlastností

Třídy mohou mít tyto skupiny **vlastností**:

- Metody - procedury/funkce, které pracují (především) s objekty této třídy
- Proměnné - pojmenované datové prvky (hodnoty) uchovávané v každém objektu této třídy


Vlastnosti jsou ve třídě "schované", tzv. **zapouzdřené** (encapsulated)

Třída připomíná pascalský záznam (record), ten však zapouzdřuje jen proměnné, nikoli metody.



Dědičnost (alespoň v javovém smyslu) znamená, že dceřinná třída (podtřída, potomek)

- má *všechny* vlastnosti (metody, proměnné) nadtřídy
- + vlastnosti uvedené přímo v deklaraci podtřídy

## Příklad

Cíl: vylepšit třídu `Ucet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Ucet>]

Postup:

1. Zdokonalíme náš příklad s účtem tak, aby si účet "hlídal", kolik se z něj převádí peněz
2. Zdokonalenou verzi třídy `Ucet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Ucet>] nazveme `KontokorentniUcet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?KontokorentniUcet>]

### Příklad 3.1. Příklad kompletního zdrojového kódu třídy

ke stažení zde  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/banka/KontokorentniUcet.java>]

```
public class KontokorentniUcet extends Ucet {
    // double zustatek; znovu neuvádíme
    // ... zdědí se z nadtřídy/předka "Ucet"







    // kolik mohu "jít do mínusu"
    double povolenyKontokorent;

    public void pridej(double castka) {
        if (zustatek + povolenyKontokorent + castka >= 0) {
            // zavoláme původní "neopatrnou" metodu
            super.pridej(castka);
        } else {
            System.err.println("Nelze odebrat částku " + (-castka));
        }
    }

    // public void vypisZustatek() ... zdědí se
    // public void prevedNa(Ucet u, double castka) ... zdědí se
    // ... předpokládáme, že v třídě "Ucet" používáme variantu:
    // pridej(-castka);
    // u.pridej(castka);
    // } }
```

Vzorový zdroják sám o sobě nepůjde přeložit, protože nemáme třídu, na níž závisí. Celý kód vystavím až po kontrole příslušných úloh.

## Příklad - co tam bylo nového

- Klíčové slovo `extends`  [<http://www.instantweb.com/foldoc/foldoc.cgi?extends>] - značí, že třída `KontokorentniUcet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?KontokorentniUcet>] je potomkem/podtřídou/rozšířením/dceřinnou třídou (*subclass*) třídy `Ucet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Ucet>].
- Konstrukce `super.metoda(...);`  [[http://www.instantweb.com/foldoc/foldoc.cgi?super.metoda\(...\);](http://www.instantweb.com/foldoc/foldoc.cgi?super.metoda(...);)] značí, že je volána metoda rodičovské třídy/předka/nadtřídy (*superclass*). *Kdyby se nevolala překrytá metoda, `super`*  [<http://www.instantweb.com/foldoc/foldoc.cgi?super>] by se neuvádělo.
- Větvění `if() {...} else {...}`  [[http://www.instantweb.com/foldoc/foldoc.cgi?if\(\) {...} else {...}](http://www.instantweb.com/foldoc/foldoc.cgi?if() {...} else {...})] - složené závorky se používají k uzavření příkazů do sekvence - ve smyslu pascalského `begin/end`.

## Přístupová práva (viditelnost)

### Přístupová práva

Přístup ke třídám i jejím prvkům lze (podobně jako např. v C++) regulovat:

- Přístupem se rozumí jakékoli použití dané třídy, prvku...
- Omezení přístupu je kontrolováno hned při překladu -> není-li přístup povolen, nelze program ani přeložit.
- Tímto způsobem lze regulovat přístup staticky, mezi celými třídami, nikoli pro jednotlivé objekty
- Jiný způsob zabezpečení představuje tzv. *security manager*, který lze aktivovat při spuštění JVM.




## Granularita omezení přístupu

Přístup je v Javě regulován *jednotlivě po prvcích*

ne jako v C++ po blocích

Omezení přístupu je určeno uvedením jednoho z tzv. *modifikátoru přístupu* (*access modifier*) nebo naopak *neuvedením žádného*.

## Typy omezení přístupu

- Existují čtyři možnosti:
  - `public`  [<http://www.instantweb.com/foldoc/foldoc.cgi?public>] = veřejný
  - `protected`  [<http://www.instantweb.com/foldoc/foldoc.cgi?protected>] = chráněný
  - modifikátor neuveden* = říká se *lokální v balíku* nebo *chráněný v balíku* nebo "přátelský"
  - `private`  [<http://www.instantweb.com/foldoc/foldoc.cgi?private>] = soukromý

## Kde jsou která omezení aplikovatelná?

Pro *třídy*:

- veřejné - `public`
- neveřejné - lokální v balíku

Pro *vlastnosti tříd* = proměnné/metody:


- veřejné - `public`
- chráněné - `protected`
- neveřejné - lokální v balíku
- soukromé - `private`

**Příklad** - `public`   
**[<http://www.instantweb.com/foldoc/foldoc.cgi?public>]**

`public`  [<http://www.instantweb.com/foldoc/foldoc.cgi?public>] => přístupné odevšad

```
public class Ucet {  
    ...  
}
```


třída

`Ucet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Ucet>]

je veřejná = lze např.

- vytvořit objekt typu Ucet  [http://www.instantweb.com/foldoc/foldoc.cgi?Ucet] i v metodě jiné třídy
- deklarovat podtřídu třídy Ucet  [http://www.instantweb.com/foldoc/foldoc.cgi?Ucet] ve stejném i jiném balíku

## Příklad - **protected** [http://www.instantweb.com/foldoc/foldoc.cgi?protected]

**protected**  [http://www.instantweb.com/foldoc/foldoc.cgi?protected] => přístupné jen z *podtříd* a *ze tříd stejného balíku*



```
public class Ucet {  
    // chráněná proměnná  
    protected float povolenyKontokorent;  
}
```

používá se jak pro metody (velmi často), tak pro proměnné (méně často)

## Příklad - přátelský


*lokální v balíku* = *přátelský* => přístupné jenze *tříd stejného balíku*, už ale ne z podtříd, jsou-li v jiném balíku

```
public class Ucet {  
    Date created; // přátelská proměnná  
}
```

- používá se spíše u proměnných než metod, ale dost často se vyskytuje z lenosti programátora, kterému se nechce psát **protected**  [http://www.instantweb.com/foldoc/foldoc.cgi?protected]
- osobně moc nedoporučuji, protože svazuje přístupová práva s organizací do balíků (-> a ta se může přece jen měnit častěji než např. vztah nadtřída-podtřída  [http://www.instantweb.com/foldoc/foldoc.cgi?nadtřída-podtřída].)
- Mohlo by mít význam, je-li práce rozdělena na více lidí na jednom balíku pracuje jen jeden člověk - pak si může přátelským přístupem chránit své neveřejné prvky/třídy -> nesmí ovšem nikdo jiný chtít mé třídy rozšiřovat a používat přitom přátelské prvky.
- Používá se relativně často pro neveřejné třídy definované v jednom zdrojovém souboru se třídou veřejnou.






## Příklad - private

`private`  [http://www.instantweb.com/foldoc/foldoc.cgi?private] => přístupné jen v rámci třídy, ani v podtřídách - používá se častěji pro proměnné než metody

označením `private` prvek *neviditelné i případným podtřídám!*

```
public class Ucet {  
    private String majitel;  
    ...  
}
```

- proměnná `majitel`  [http://www.instantweb.com/foldoc/foldoc.cgi?majitel] je soukromá = nelze k ní přímo přistoupit ani v podtřídě - je tedy třeba zpřístupnit proměnnou pro "vnější" potřeby jinak, např.
- přístupovými metodami `setMajitel(String m)`  [http://www.instantweb.com/foldoc/foldoc.cgi?setMajitel(String m)] a `String getMajitel()`  [http://www.instantweb.com/foldoc/foldoc.cgi?String getMajitel()]

## Když si nevíte rady

Nastavení přístupových práv k třídě pomocí modifikátorů se děje na úrovni tříd, tj. vztahuje se pak na *všechny objekty příslušné třídy* i na její *statické vlastnosti* (proměnné, metody) atd.




Nastavení musí vycházet z povahy dotyčné proměnné či metody.

Nevíme-li si rady, jaká práva přidělit, řídíme se následujícím:

- metoda by měla být `public`  [http://www.instantweb.com/foldoc/foldoc.cgi?public], je-li užitečná i mimo třídu či balík - "navenek"
- jinak `protected`  [http://www.instantweb.com/foldoc/foldoc.cgi?protected]
- máme-li záruku, že metoda bude v případných podtřídách nepotřebná, může být `private`  [http://www.instantweb.com/foldoc/foldoc.cgi?private] - *ale kdy tu záruku máme???*
- proměnná by měla být `private`  [http://www.instantweb.com/foldoc/foldoc.cgi?private], nebo `protected`  [http://www.instantweb.com/foldoc/foldoc.cgi?protected], je-li potřeba přímý přístup v podtřídě
- téměř nikdy bychom neměli deklarovat proměnné jako `public` 


[<http://www.instantweb.com/foldoc/foldoc.cgi?public>]

## Přístupová práva a umístění deklarací do souborů


- Třídy deklarované jako veřejné (`public`  [<http://www.instantweb.com/foldoc/foldoc.cgi?public>]) musí být umístěné do souborů s názvem totožným s názvem třídy (+přípona `.java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.java>]) i na systémech Windows (vč. velikosti písmen)
- kromě takové třídy však může být v tomto souboru i libovolný počet deklarací neveřejných tříd
- `private`  [<http://www.instantweb.com/foldoc/foldoc.cgi?private>] nemají význam, ale *přátelské* ano

## Organizace tříd do balíků

### Zápis třídy do zdrojového souboru

Soubor `Zivocich.java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Zivocich.java>] bude obsahovat (pozor na velká/malá písmena - v obsahu i názvu souboru):

```
public class Zivocich {  
    ... popis vlastností (proměnných, metod...) živočicha ...  
}
```

`public`  [<http://www.instantweb.com/foldoc/foldoc.cgi?public>] značí, že třída je "veřejně" použitelná, tj. i mimo balík

## Organizace tříd do balíků

Třídy zorganizujeme do balíků.

V balíku jsou vždy umístěny *související* třídy.

Co znamená související?



- třídy, jejichž **objekty spolupracují** - *člověk, úřad*
- třídy na podobné **úrovni abstrakce** - *chovatel, domácí zvíře*
- třídy ze **stejné části reality** - *chovatel psů, pes*

## Balíky

Balíky obvykle organizujeme do hierarchií, např.:


- `svet`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet>]
- `svet.chovatelstvi`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi>]
- `svet.chovatelstvi.psi`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.psi>]
- `svet.chovatelstvi.morcata`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.morcata>]



Neplatí však, že by

- třídy "dceřinného" balíku (např. `svet.chovatelstvi.psi`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.psi>])
- byly zároveň třídami balíku "rodičovského" (`svet.chovatelstvi`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi>]))!!!

Hierarchie balíků má tedy význam spíše pro srozumitelnost a logické členění.


## Příslušnost třídy k balíku

Deklarujeme ji syntaxí: **package** **názevbalíku;**   
[<http://www.instantweb.com/foldoc/foldoc.cgi?package=názevbalíku;>]

- Uvádíme obvykle jako *první* deklaraci v zdrojovém souboru;
- Příslušnost k balíku musíme současně *potvrdit správným umístěním* zdrojového souboru do adresářové struktury;
- např. zdrojový soubor třídy `Pes`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Pes>] umístíme do podadresáře `svet\chovatelstvi\psi`  [<http://www.instantweb.com/foldoc/foldoc.cgi?svet\chovatelstvi\psi>]
- Neuvedeme-li příslušnost k balíku, stane se třída součástí **implicitního balíku** - to však nelze pro jakékoli větší a/nebo znovupoužívané třídy či dokonce programy doporučit a zde nebude tolerováno!

## Deklarace

**import**

**NázevTřídy** 

## [<http://www.instantweb.com/foldoc/foldoc.cgi?import> **NázevTřídy**]

Deklarace import nesouvisí s děděním, ale s organizací tříd programu do balíků:

- Umožní odkazovat se *v rámci kódu jedné třídy na ostatní třídy*
- Syntaxe: `import názevtřídy;` [[G](http://www.instantweb.com/foldoc/foldoc.cgi?import)] [<http://www.instantweb.com/foldoc/foldoc.cgi?import> názevtřídy;]
- kde *názevtřídy* je uveden včetně názvu balíku
- Pišeme obvykle ihned po deklaraci příslušnosti k balíku (`package názevbalíku;` [[G](http://www.instantweb.com/foldoc/foldoc.cgi?package)] [<http://www.instantweb.com/foldoc/foldoc.cgi?package> názevbalíku;])

Import není nutné deklarovat mezi třídami téhož balíku!

## **Deklarace** `import` `názevbalíku.*` [[G](http://www.instantweb.com/foldoc/foldoc.cgi?import)] [<http://www.instantweb.com/foldoc/foldoc.cgi?import> **názevbalíku.\***]

Pak lze používat všechny třídy z uvedeného balíku

Doporučuje se "import s hvězdičkou" nepoužívat:

- jinak nevíme nikdy s jistotou, ze kterého balíku se daná třída použila;
- i profesionálové to však někdy používají :-)
- lze tolerovat tam, kde používáme z určitého balíku většinu tříd;
- v tomto úvodním kurzu většinou tolerovat nebudeme!

"Hvězdičkou" *nezpřístupníme třídy z podbalíků*, např.

- `import svet.*` [[G](http://www.instantweb.com/foldoc/foldoc.cgi?import)] [<http://www.instantweb.com/foldoc/foldoc.cgi?import> svet.\*] *nezpřístupní* třídu `svet.chovatelstvi.Chovatel` [[G](http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.Chovatel)] [<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.Chovatel>]

---

# Kapitola 4. Datové typy: primitivní, objektové, pole. Výrazy.

## Úvod k datovým typům v Javě

- Primitivní vs. objektové typy
- Kategorie primitivních typů: integrální, boolean, čísla s pohyblivou řádovou čárkou
- Pole: deklarace, vytvoření, naplnění, přístup k prvkům, rozsah indexů

## Primitivní vs. objektové datové typy - opakování

Java striktně rozlišuje mezi hodnotami

- **primitivních datových typů** (čísla, logické hodnoty, znaky) a
- **objektových typů** (řetězce a všechny uživatelem definované [tj. vlastní] typy-třídy)

Základní rozdíl je v práci s proměnnými:

- proměnné primitivních typů *přímo obsahují danou hodnotu*, zatímco
- proměnné objektových typů obsahují pouze *odkaz na příslušný objekt*

Důsledek -> **dvě objektové proměnné** mohou nést odkaz na **tentýž objekt**


## Přiřazení proměnné primitivního typu - opakování

- Příklad:

```
float a = 1.23456;  
float b = a;  
a += 2;
```

## Přiřazení objektové proměnné - opakování

- Příklad, deklarujeme třídu

Cislo  [<http://www.instantweb.com/foldoc/foldoc.cgi?Cislo>]

takto:

```
public class Cislo {  
    private float hodnota;  
    public Cislo(float h) {  
        hodnota = h;  
    }  
    public void zvysO(float kolik) {  
        hodnota += kolik;  
    }  
    public void vypis() {  
        System.out.println(hodnota);  
    }  
}
```

- nyní ji použijeme:

```
Cislo c1 = new Cislo(1.23456);  
Cislo c2 = c1;  
c1.zvysO(2);  
c1.vypis();  
c2.vypis();
```

dostaneme:


```
3.23456  
3.23456
```

## Primitivní datové typy









### Primitivní datové typy

Proměnné těchto typů nesou **elementární**, z hlediska Javy **atomické**, dále **nestrukturované** hodnoty.

Deklarace takové proměnné (kdekoli) způsobí:




1. rezervování příslušného paměťového prostoru (např. pro hodnotu `int`  [<http://www.instantweb.com/foldoc/foldoc.cgi?int>] čtyři bajty)
2. zpřístupnění (pojmenování) tohoto prostoru identifikátorem proměnné

V Javě existují tyto skupiny primitivních typů:

1. **integrální typy** (obdoba ordinálních typů v Pascalu) - zahrnují typy *celočíselné* (byte   
[http://www.instantweb.com/foldoc/foldoc.cgi?byte], short   
[http://www.instantweb.com/foldoc/foldoc.cgi?short], int   
[http://www.instantweb.com/foldoc/foldoc.cgi?int] a long   
[http://www.instantweb.com/foldoc/foldoc.cgi?long]) a typ char   
[http://www.instantweb.com/foldoc/foldoc.cgi?char];
2. typy **čísels** s **pohyblivou řádovou čárkou** (float   
[http://www.instantweb.com/foldoc/foldoc.cgi?float] a double   
[http://www.instantweb.com/foldoc/foldoc.cgi?double])
3. typ **logických hodnot** (boolean  [http://www.instantweb.com/foldoc/foldoc.cgi?boolean]).



## Integrální typy - celočíselné


V Javě jsou celá čísla vždy interpretována jako znaménková

"Základním" celočíselným typem je 32bitový int   
[http://www.instantweb.com/foldoc/foldoc.cgi?int] s rozsahem -2 147 483 648   
[http://www.instantweb.com/foldoc/foldoc.cgi?-2 147 483 648] až 2 147 483 647   
[http://www.instantweb.com/foldoc/foldoc.cgi?2147483647]


větší rozsah (64 bitů) má long  [http://www.instantweb.com/foldoc/foldoc.cgi?long], cca +/-  
9\*10<sup>18</sup>  [http://www.instantweb.com/foldoc/foldoc.cgi?+/- 9\*10<sup>18</sup>]


menší rozsah mají

- short  [http://www.instantweb.com/foldoc/foldoc.cgi?short] (16 bitů), tj. -32768..32767
- byte  [http://www.instantweb.com/foldoc/foldoc.cgi?byte] (8 bitů), tj. -128..127

Pro celočíselné typy existují (stejně jako pro floating-point typy) konstanty - *minimální a maximální hodnoty* příslušného typu. Tyto konstanty mají název vždy Typ.MIN\_VALUE   
[http://www.instantweb.com/foldoc/foldoc.cgi? Typ.MIN\_VALUE ], analogicky MAX... Viz např.  
Minimální a maximální hodnoty  
[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/hodnoty/MinMaxHodnoty.java]

## Integrální typy - "char"

`char`  [<http://www.instantweb.com/foldoc/foldoc.cgi?char>] představuje jeden 16bitový znak v kódování UNICODE

Konstanty typu `char`  [<http://www.instantweb.com/foldoc/foldoc.cgi?char>] zapisujeme

- v apostrofech - 'a', 'Ř'
- pomocí escape-sekvencí - \n (konec řádku) \t (tabulátor)
- hexadecimálně - \u0040 (totéž, co 'a')
- oktalově - \127


## Typ `char` [<http://www.instantweb.com/foldoc/foldoc.cgi?char>] - kódování

Java vnitřně kóduje znaky a řetězce v UNICODE, pro vstup a výstup je třeba použít některou za serializací (převodu) UNICODE na sekvence bajtů:

- např. vícebajtová kódování UNICODE: **UTF-8** a UTF-16
- osmibitová kódování ISO-8859-x, Windows-125x a pod.

Problém může nastat při interpretaci kódování znaků národních abeced přímo ve zdrojovém textu programu.


Ve zdroj. textu správně napsaného javového vícejazyčného programu by žádné národní znaky VŮBEC neměly vyskytovat.


Je vhodné umístit je do speciálních souborů tzv. *zdrojů* (v Javě objekty třídy `java.util.ResourceBundle` 

[<http://www.instantweb.com/foldoc/foldoc.cgi?java.util.ResourceBundle>]).

## Čísla s pohyblivou řádovou čárkou

Kódována podle ANSI/IEEE 754-1985

- `float`  [<http://www.instantweb.com/foldoc/foldoc.cgi?float>] - 32 bitů
- `double`  [<http://www.instantweb.com/foldoc/foldoc.cgi?double>] - 64 bitů

Možné zápisy literálů typu `float`  [<http://www.instantweb.com/foldoc/foldoc.cgi?float>] (klasický i



semilogaritmický tvar) - povšimněte si "f" za číslem - je u float nutné!:

```
float f = -.777f, g = 0.123f, h = -4e6f, 1.2E-15f;
```

[[http://www.instantweb.com/foldoc/foldoc.cgi?float f = -.777f, g = 0.123f, h = -4e6f, 1.2E-15f;](http://www.instantweb.com/foldoc/foldoc.cgi?float+f=-.777f,g=0.123f,h=-4e6f,1.2E-15f;)]

double [<http://www.instantweb.com/foldoc/foldoc.cgi?double>]: tentýž zápis, ovšem bez "f" za konstantou!, s větší povolenou přesností a rozsahem

## Vestavěné konstanty s pohyblivou řádovou čárkou

Kladné a záporné nekonečno:

- `Float.POSITIVE_INFINITY` [[http://www.instantweb.com/foldoc/foldoc.cgi?Float.POSITIVE\\_INFINITY](http://www.instantweb.com/foldoc/foldoc.cgi?Float.POSITIVE_INFINITY) ], totéž s `NEGATIVE` [<http://www.instantweb.com/foldoc/foldoc.cgi?NEGATIVE>]...
- totéž pro `Double` [<http://www.instantweb.com/foldoc/foldoc.cgi?Double>]
- obdobně existují pro oba typy konstanty uvádějící rozlišení daného typu - `MIN_VALUE` [[http://www.instantweb.com/foldoc/foldoc.cgi? MIN\\_VALUE](http://www.instantweb.com/foldoc/foldoc.cgi?MIN_VALUE) ], podobně s `MAX` [<http://www.instantweb.com/foldoc/foldoc.cgi?MAX>]...

Konstanta `NaN` [<http://www.instantweb.com/foldoc/foldoc.cgi?NaN> ] - *Not A Number*

Viz také Minimální a maximální hodnoty  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/hodnoty/MinMaxHodnoty.java>]

## Typ logických hodnot - *boolean*

Přípustné hodnoty jsou `false` [<http://www.instantweb.com/foldoc/foldoc.cgi?false>] a `true` [<http://www.instantweb.com/foldoc/foldoc.cgi?true>].

Na rozdíl od Pascalu na nich *není* definováno uspořádání.

## Typ `void` [<http://www.instantweb.com/foldoc/foldoc.cgi?void>]

Význam podobný jako v C/C++.

Není v pravém slova smyslu datovým typem, nemá žádné hodnoty.

Označuje "prázdný" typ pro sdělení, že určitá metoda *nevrací žádný výsledek*.

## Pole

## Pole v Javě

Pole v Javě je speciálním **objektem**

Můžeme mít pole jak primitivních, tak objektových hodnot

- pole primitivních hodnot tyto **hodnoty obsahuje**
- pole objektů obsahuje **odkazy na objekty**

Kromě pole v Javě existují i jiné objekty na ukládání více hodnot - tzn. kontejnery, viz dále

Syntaxe deklarace



```
typhodnoty [] jménopole  [http://www.instantweb.com/foldoc/foldoc.cgi?typhodnoty []  
jménopole]
```



### Poznámka

na rozdíl od C/C++ nikdy neuvádíme při deklaraci počet prvků pole - ten je podstatný až při **vytvoření objektu pole**

Syntaxe přístupu k prvkům *jménopole[indexprvku]* Používáme

- jak pro **přiřazení** prvku do pole: `jménopole[indexprvku] = hodnota;`   
[http://www.instantweb.com/foldoc/foldoc.cgi? jménopole[indexprvku] = hodnota;]
- tak pro **čtení** hodnoty z pole `proměnná = jménopole[indexprvku];`   
[http://www.instantweb.com/foldoc/foldoc.cgi? proměnná = jménopole[indexprvku];]

Syntaxe vytvoření *objektu* pole: jako u jiného objektu - voláním konstruktoru:


*jménopole = new typhodnoty[ početprvků ];* nebo vzniklé pole rovnou naplníme hodnotami/odkazy

*jménopole = new typhodnoty[] {prvek1, prvek2, ...};*

## Pole (2)



Pole je objekt, je třeba ho před použitím nejen **deklarovat**, ale i **vytvořit**:

```
Clovek[] lidi;  
lidi = new Clovek[5];
```

```
Clovek  [http://www.instantweb.com/foldoc/foldoc.cgi?Clovek]
```

Nyní můžeme pole naplnit:

```
lidi[0] = new Clovek("Václav Klaus", Clovek.MUZ);  
lidi[1] = new Clovek("Libuše Benešová", Clovek.ZENA);  
  
lidi[0].vypisInfo(); lidi[1].vypisInfo();
```

- Nyní jsou v poli `lidi`  [<http://www.instantweb.com/foldoc/foldoc.cgi?lidi>] naplněny první dva prvky odkazy na objekty.
- Zbylé prvky zůstaly naplněny prázdnými odkazy `null`  [<http://www.instantweb.com/foldoc/foldoc.cgi?null>].

## Pole - co když deklarujeme, ale nevytvoříme?

Co kdybychom pole pouze deklarovali a nevytvořili:

```
Clovek[] lidi;  
lidi[0] = new Clovek("Václav Klaus", Clovek.MUZ);
```

Toto by skončilo s běhovou chybou "NullPointerException", pole neexistuje, nelze do něj tudíž vkládat prvky!


Pokud tuto chybu uděláme v rámci metody:

```
public class Pokus {  
    public static void main(String args[]) {  
        String[] pole;  
        pole[0] = "Neco";  
    }  
}
```

překladač nás varuje:

```
Pokus.java:4: variable pole might not have been  
    initialized pole[0] = "Neco"; ^ 1 error
```

Pokud ovšem

`pole`  [<http://www.instantweb.com/foldoc/foldoc.cgi?pole>]

bude proměnnou objektu/třídy:

```
public class Pokus {
```

```
static String[] pole;  
public static void main(String args[]) {  
    pole[0] = "Neco";  
}  
}
```

Překladač chybu neodhalí a po spuštění se objeví:

```
Exception in thread "main"  
java.lang.NullPointerException at Pokus.main(Pokus.java:4)
```

## Pole - co když deklarujeme, vytvoříme, ale nenaplníme?

Co kdybychom pole deklarovali, vytvořili, ale nenaplnili příslušnými prvky:

```
Clovek[] lidi;  
lidi = new Clovek[5];  
lidi[0].vypisInfo();
```

Toto by skončilo také s běhovou chybou *NullPointerException*:

- pole existuje, má pět prvků, ale první z nich je prázdný, nelze tudíž volat jeho metody (resp. vůbec používat jeho vlastnosti)!

## Kopírování polí

V Javě obecně přiřazení proměnné objektového typu vede pouze k **duplikaci odkazu**, **nikoli** celého odkazovaného **objektu**.

Nejinak je tomu u polí, tj.:

```
Clovek[] lidi2;  
lidi2 = lidi1;
```

V proměnné *lidi2* je nyní odkaz na stejné pole jako je *lidi1*.

Zatímco, provedeme-li vytvoření nového pole + *arraycopy*, pak *lidi2* obsahuje duplikát/klon/kopii původního pole.

```
Clovek[] lidi2 = new Clovek[5];  
System.arraycopy(lidi, 0, lidi2, 0, lidi.length);
```

viz též Dokumentace API třídy "System" [<http://java.sun.com/j2se/1.4/docs/api/java/lang/System.html>]



## Poznámka

Samozřejmě bychom mohli kopírovat prvky ručně, např. pomocí `for` [http://www.instantweb.com/foldoc/foldoc.cgi?for] cyklu, ale volání `System.arraycopy` je zaručeně nejrychlejší a přitom stále platformově nezávislou metodou, jak kopírovat pole.

Také `arraycopy` však do cílového pole zduplikuje jen **odkazy na objekty**, nevytvoří kopie objektů!

# Operátory a výrazy

- Operátory v Javě: aritmetické, logické, relační, bitové
- Ternární operátor podmíněného výrazu
- Typové konverze
- Operátor zřetězení

## Aritmetické

|                                                                                              |                                                                 |
|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| <code>+</code> [http://www.instantweb.com/foldoc/foldoc.cgi?+],                              | <code>-</code> [http://www.instantweb.com/foldoc/foldoc.cgi?-], |
| <code>[http://www.instantweb.com/foldoc/foldoc.cgi?*-]</code> ,                              | <code>*</code> [http://www.instantweb.com/foldoc/foldoc.cgi?*], |
| <code>[http://www.instantweb.com/foldoc/foldoc.cgi?*/]</code> ,                              | <code>/</code> [http://www.instantweb.com/foldoc/foldoc.cgi?/], |
| <code>[http://www.instantweb.com/foldoc/foldoc.cgi?%]</code> (zbytek po celočíselném dělení) | <code>%</code> [http://www.instantweb.com/foldoc/foldoc.cgi?%]  |

## Logické

*logické součiny (AND):*


- `&` [http://www.instantweb.com/foldoc/foldoc.cgi? & ] (*nepodmíněný* - vždy se vyhodnotí oba operandy),
- `&&` [http://www.instantweb.com/foldoc/foldoc.cgi? && ] (*podmíněný* - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

*logické součty (OR):*

- `|` [http://www.instantweb.com/foldoc/foldoc.cgi? | ] (*nepodmíněný* - vždy se vyhodnotí oba operandy),
- `||` [http://www.instantweb.com/foldoc/foldoc.cgi? || ] (*podmíněný* - líné vyhodnocování - dru-





hý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

*negace* (NOT):



- !  [http://www.instantweb.com/foldoc/foldoc.cgi? ! ]

## Relační (porovnávací)

Tyto lze použít na porovnávání primitivních hodnot:

- <  [http://www.instantweb.com/foldoc/foldoc.cgi? < ], <=   
[http://www.instantweb.com/foldoc/foldoc.cgi? <= ], >=   
[http://www.instantweb.com/foldoc/foldoc.cgi? >= ], >   
[http://www.instantweb.com/foldoc/foldoc.cgi? > ]

Test na rovnost/nerovnost lze použít na porovnávání primitivních hodnot i objektů:

- ==  [http://www.instantweb.com/foldoc/foldoc.cgi? == ], !=   
[http://www.instantweb.com/foldoc/foldoc.cgi? != ]

Upozornění:




- pozor na porovnávání objektů: == vrací true jen při rovnosti odkazů, tj. jsou-li objekty identické. Rovnost obsahu (tedy "rovnocennost") objektů se zjišťuje voláním metody `o1.equals(Object o2)`
- pozor na srovnávání floating-points čísel na rovnost: je třeba počítat s chybami zaokrouhlení; místo porovnání na přesnou rovnost raději použijeme jistou toleranci: `abs(expected-actual) < delta`

## Bitové

*Bitové:*

- součin &  [http://www.instantweb.com/foldoc/foldoc.cgi? & ]
- součet |  [http://www.instantweb.com/foldoc/foldoc.cgi? | ]
- exkluzivní součet (XOR) ^  [http://www.instantweb.com/foldoc/foldoc.cgi? ^ ] (znak "stříška")
- negace (bitwise-NOT) ~  [http://www.instantweb.com/foldoc/foldoc.cgi? ~ ] (znak "tilda")

Posuny:

- vlevo << [http://www.instantweb.com/foldoc/foldoc.cgi?<<] o stanovený počet bitů
- vpravo >> [http://www.instantweb.com/foldoc/foldoc.cgi?>>] o stanovený počet bitů s respektováním znaménka
- vpravo >>> [http://www.instantweb.com/foldoc/foldoc.cgi?>>>] o stanovený počet bitů bez respektování znaménka


Dále                      viz                      např.                      Bitové                      operátory  
[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/operators/Bitove.java]

## Operátor                      podmíněného                      výrazu                      ?                      : [http://www.instantweb.com/foldoc/foldoc.cgi?? :]

Jediný ternární operátor

Platí-li první operand (má hodnotu `true`  [http://www.instantweb.com/foldoc/foldoc.cgi?true]) ->

- výsledkem je hodnota druhého operandu
- jinak je výsledkem hodnota třetího operandu


Typ prvního operandu musí být `boolean`  [http://www.instantweb.com/foldoc/foldoc.cgi?boolean],  
typ druhého a třetího musí být přiřaditelné do výsledku.

## Operátory typové konverze (přetypování)


- Podobně jako v C/C++
- Píše se (*typ*)*hodnota*, např. (Clovek)o, kde o byla proměnná deklarovaná jako Object.
- Pro objektové typy se ve skutečnosti nejedná o žádnou konverzi spojenou se změnou obsahu objektu, nýbrž pouze o potvrzení, že běhový typ objektu je požadovaného typu - např. (viz výše) že o je typu Clovek.
- Naproti tomu u primitivních typů se jedná o úpravu hodnoty - např. int přetypujeme na short a „oreže“ se tím rozsah.





## Operátor                      zřetězení                      + [http://www.instantweb.com/foldoc/foldoc.cgi?+]

Výsledkem je vždy řetězec, ale argumenty mohou být i jiných typů, např.

sekvence `int i = 1; System.out.println("promenna i="+i);`   
[[http://www.instantweb.com/foldoc/foldoc.cgi?int i = 1; System.out.println\("promenna i="+i\);](http://www.instantweb.com/foldoc/foldoc.cgi?int i = 1; System.out.println("promenna i=)] je v po-  
řádku

s řetězcovou konstantou se spojí řetězcová podoba dalších argumentů (např. čísla).

Pokud je argumentem zřetězení odkaz na objekt `o`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?o>] ->

- je-li `o == null`  [<http://www.instantweb.com/foldoc/foldoc.cgi?o == null>] -> použije se řetě-  
zec `null`  [<http://www.instantweb.com/foldoc/foldoc.cgi?null>]
- je-li `o != null`  [<http://www.instantweb.com/foldoc/foldoc.cgi?o != null>] -> použije se hodno-  
ta vrácená metodou `o.toString()`   
[[http://www.instantweb.com/foldoc/foldoc.cgi?o.toString\(\)](http://www.instantweb.com/foldoc/foldoc.cgi?o.toString())] (tu lze překrýt a dosáhnout tak očekáva-  
ného řetězcového výstupu)

## Priority operátorů a vytváření výrazů





nejvyšší prioritu má násobení, dělení, nejnižší přiřazení

## Porovnávání objektů

- Porovnávání primitivních hodnot a objektů
- metody `equals` a `hashCode`

## Relační (porovnávací)



Tyto lze použít na porovnávání primitivních hodnot:

- `<`  [<http://www.instantweb.com/foldoc/foldoc.cgi?<>], `<=`   
[[<=](http://www.instantweb.com/foldoc/foldoc.cgi?<=)], `>=`   
[[>=](http://www.instantweb.com/foldoc/foldoc.cgi?>=)], `>`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?>>]

Test na rovnost/nerovnost lze použít na porovnávání primitivních hodnot i objektů:

-



 [http://www.instantweb.com/foldoc/foldoc.cgi? == ], !=   
[http://www.instantweb.com/foldoc/foldoc.cgi? != ]

Upozornění:

- pozor na porovnávání objektů: == vrací true jen při rovnosti odkazů, tj. jsou-li objekty identické. Rovnost obsahu (tedy "rovnocennost") objektů se zjišťuje voláním metody `o1.equals(Object o2)`
- pozor na srovnávání floating-points čísel na rovnost: je třeba počítat s chybami zaokrouhlení; místo porovnání na přesnou rovnost raději použijeme jistou toleranci: `abs(expected-actual) < delta`

## Porovnávání objektů

Použití ==

- Porovnáme-li dva objekty (tzn. odkazy na objekty) prostřednictvím operátoru == dostaneme rovnost jen v případě, jedná-li se o dva odkazy na tentýž objekt - tj. dva *totožné* objekty.
- Jedná-li se o dva obsahově stejné objekty existující samostatně, pak == vrátí false.

Chceme-li (intuitivně) chápat rovnost objektů podle obsahu, tj.

- dva objekty jsou *rovné* (*rovnocenné*, nikoli *totožné*), mají-li stejný obsah, pak
- musíme pro danou třídu překrýt metodu `equals`, která musí vrátit true, právě když se *obsah* výchozího a srovnávaného objektu rovná.
- Fungování `equals` lze srovnat s porovnáváním dvou databázových záznamů podle primárního klíče.
- Nepřekryjeme-li `equals`, funguje původní `equals` přísným způsobem, tj. *rovné si budou jen totožné objekty*.

## Porovnávání objektů - příklad

Příklad: objekt třídy `Clovek` nese informace o člověku. Dva objekty položíme stejné (rovnocenné), nejsou-li stejná příjmení:

### Obrázek 4.1. Dva lidi jsou stejní, mají-li stejná příjmení

```
public class Clovek implements Comparable {
    String jmeno, prijmeni;
    public Clovek (String j, String p) {
        jmeno = j;
```

```
        prijmeni = p;
    }
    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek)o;
            return prijmeni.equals(c.prijmeni);
        } else
            throw new IllegalArgumentException(
                "Nelze porovnat objekt typu Clovek s objektem jineho typu");
    }
}
```

Méně agresivní verze by nemusela při porovnávání s jiným objektem než Clovek vyhodit výjimku, pouze vrátit false.

## Metoda hashCode

Jakmile u třídy překryjeme metodu equals, měli bychom současně překrýt objektů i metodu hashCode:

- hashCode vrací celé číslo (int) „co nejlépe“ charakterizující obsah objektu, tj.
- pro dva stejné (equals) objekty *musí vždy vrátit stejnou hodnotu*.
- Pro dva obsahově různé objekty by hashCode naopak měl vracet různé hodnoty (ale není to stoprocentně nezbytné a ani nemůže být vždy splněno). Metoda hashCode totiž nemůže vždy být prostá.

## Metoda hashCode - příklad

V těle hashCode s oblibou „přehráváme“ (delegujeme) řešení na volání hashCode jednotlivých složek objektu - a to těch, které figurují v equals:

### Obrázek 4.2. Třída Clovek s metodami equals a hashCode

```
public class Clovek implements Comparable {
    String jmeno, prijmeni;
    public Clovek (String j, String p) {
        jmeno = j;
        prijmeni = p;
    }
    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek)o;
            return prijmeni.equals(c.prijmeni);
        } else
```

```
        throw new IllegalArgumentException(  
            "Nelze porovnat objekt typu Clovek s objektem jineho typu");  
    }  
    public int hashCode() {  
        return prijmeni.hashCode();  
    }  
}
```

---

# Kapitola 5. Řídicí struktury: větvení, cykly.

## Příkazy a řídicí struktury v Javě

- Příkazy v Javě
- Přiřazení
- Volání metod a návrh z nich
- Řídicí příkazy (větvení, cykly)

## Příkazy v Javě

### Příkazy v Javě

V Javě máme následující příkazy:

- Přiřazovací příkaz = a jeho modifikace (kombinované operátory jako je += apod.)
- Řízení toku programu (větvení, cykly)
- Volání metody
- Návrat z metody - příkaz return
- Příkaz je ukončen středníkem ;
- v Pascalu středník příkazy *odděluje*, v Javě (C/C++) *ukončuje*

## Přiřazení

### Přiřazení v Javě

Operátor přiřazení:

- Operátor přiřazení = (assignment)
- na levé straně musí být proměnná
- na pravé straně výraz *přiřaditelný* (assignable) do této proměnné

- Rozlišujeme přiřazení primitivních hodnot a odkazů na objekty

## Přiřazení primitivní hodnoty

Na pravé straně výraz vracející hodnotu primitivního typu

číslo, logická hodnota, znak (ale ne např. řetězec)

Na levé straně proměnná téhož typu jako přiřazovaná hodnota nebo typu širšího

např. `int [http://www.instantweb.com/foldoc/foldoc.cgi?int]` lze přiřadit do `long [http://www.instantweb.com/foldoc/foldoc.cgi?long]`

Při zužujícím přiřazení se také provede konverze, ale může dojít ke ztrátě informace

např. `int [http://www.instantweb.com/foldoc/foldoc.cgi?int]` -> `short [http://www.instantweb.com/foldoc/foldoc.cgi?short]`

Přiřazením primitivní hodnoty se hodnota zduplikuje ("opíše") do proměnné na levé straně.

## Přiřazení odkazu na objekt

Konstrukci `= [http://www.instantweb.com/foldoc/foldoc.cgi?]` lze použít i pro přiřazení do objektové proměnné:

`Zivocich z1 = new Zivocich(); [http://www.instantweb.com/foldoc/foldoc.cgi?Zivocich z1 = new Zivocich();]`

Co to udělalo?

1. vytvořilo nový objekt typu `Zivocich [http://www.instantweb.com/foldoc/foldoc.cgi?Zivocich]` (`new Zivocich() [http://www.instantweb.com/foldoc/foldoc.cgi? new Zivocich() ]`)
2. přiřadilo jej do proměnné `z1 [http://www.instantweb.com/foldoc/foldoc.cgi?z1]` typu `Zivocich [http://www.instantweb.com/foldoc/foldoc.cgi?Zivocich]`

Nyní můžeme odkaz na tentýž vytvořený objekt znovu přiřadit - do `z2 [http://www.instantweb.com/foldoc/foldoc.cgi?z2]`:

`Zivocich z2 = z1; [http://www.instantweb.com/foldoc/foldoc.cgi?Zivocich z2 = z1;]`

Proměnné `z1 [http://www.instantweb.com/foldoc/foldoc.cgi?z1]` a `z2 [http://www.instantweb.com/foldoc/foldoc.cgi?z2]` ukazují nyní na stejný objekt typu živočich!!!

Proměnné objektového typu obsahují *odkazy* (reference) na objekty, ne objekty samotné!!!

## Volání metod a návrat z nich

### Volání metody

Metoda objektu je vlastně procedura/funkce, která realizuje svou činnost primárně s proměnnými objektu.


Volání metody určitého objektu realizujeme:

*identifikaceObjektu.názevMetody(skutečné parametry)*

- **identifikaceObjektu**, jehož metodu voláme
- **.** (tečka)
- **názevMetody**, již nad daným objektem voláme
- v závorkách uvedeme *skutečné parametry* volání (záv. může být prázdná, nejsou-li parametry)

### Návrat z metody

Návrat z metody se děje:

1. Buďto automaticky posledním příkazem v těle metody
2. nebo explicitně příkazem **return** *návratová hodnota*   
[<http://www.instantweb.com/foldoc/foldoc.cgi?return> ]


způsobí ukončení provádění těla metody a návrat, přičemž může být specifikována *návratová hodnota*

typ skutečné návratové hodnoty musí korespondovat s deklarovaným typem návratové hodnoty



## Řízení toku uvnitř metod - větvení, cykly

### Řízení toku programu v těle metody




Příkaz (neúplného) větvení **if**  [<http://www.instantweb.com/foldoc/foldoc.cgi?if>]


**if** (logický výraz) **příkaz**  [<http://www.instantweb.com/foldoc/foldoc.cgi?if> (logický výraz) **příkaz**]

platí-li logický výraz (má hodnoty true), provede se příkaz

Příkaz úplného větvení **if - else**  [<http://www.instantweb.com/foldoc/foldoc.cgi?else>]   
[<http://www.instantweb.com/foldoc/foldoc.cgi?if> - ]

```
if (logický výraz)
    příkaz1
else
    příkaz2
```

platí-li logický výraz  [[http://www.instantweb.com/foldoc/foldoc.cgi? logický výraz](http://www.instantweb.com/foldoc/foldoc.cgi?logický%20výraz) ] (má hodnoty **true**  [<http://www.instantweb.com/foldoc/foldoc.cgi?true>]), provede se příkaz1   
[[http://www.instantweb.com/foldoc/foldoc.cgi? příkaz1](http://www.instantweb.com/foldoc/foldoc.cgi?příkaz1) ]

neplatí-li, provede se příkaz2  [[http://www.instantweb.com/foldoc/foldoc.cgi? příkaz2](http://www.instantweb.com/foldoc/foldoc.cgi?příkaz2) ]

Větev **else**  [<http://www.instantweb.com/foldoc/foldoc.cgi?else>] se **nemusí uvádět**

## Cyklus s podmínkou na začátku

Tělo cyklu se provádí tak dlouho, **dokud** platí podmínka

obdoba **while**  [<http://www.instantweb.com/foldoc/foldoc.cgi?while>] v Pascalu

v těle cyklu je jeden jednoduchý příkaz ...

```
while (podmínka)
    příkaz;
```

... nebo příkaz složený

```
while (podmínka) {
    příkaz1;
    příkaz2;
    příkaz3;
    ...
}
```

Tělo cyklu se nemusí provést ani jednou - pokud už hned na začátku podmínka neplatí

## Doporučení k psaní cyklů/větvení

Větvení, cykly: doporučuji vždy psát se **složeným příkazem v těle** (tj. se složenými závorkami)!!!

jinak hrozí, že se v těle větvení/cyklu z neopatrnosti při editaci objeví něco jiného, než chceme, např.:

```
while (i < a.length)
    System.out.println(a[i]); i++;
```

Provede v cyklu jen ten výpis, inkrementaci ne a program se zacyklí.

Pišme proto vždy takto:

```
while (i < a.length) {  
    System.out.println(a[i]); i++;  
}
```

## Příklad použití "while" cyklu

Dokud nejsou přečteny všechny vstupní argumenty:


```
int i = 0;  
while (i < args.length) {  
    "přečti argument args[i]"  
    i++;  
}
```


Dalším příkladem je použití **while**  [<http://www.instantweb.com/foldoc/foldoc.cgi?while>] pro realizaci celočíselného dělení se zbytkem:

Příklad:                      Celočíselné                      dělení                      se                      zbytkem  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/DeleniOdcitanim.java>]

## Cyklus s podmínkou na konci

Tělo se provádí **dokud** platí podmínka (vždy aspoň jednou)

obdoba **repeat**  [<http://www.instantweb.com/foldoc/foldoc.cgi?repeat>] v Pascalu (podmínka je ovšem *interpretována opačně*)

Relativně      málo      používaný      -      je      méně      přehledný      než      **while**   
[<http://www.instantweb.com/foldoc/foldoc.cgi?while>]

Syntaxe:

```
do {  
    příkaz1;  
    příkaz2;  
    příkaz3;  
    ...  
} while (podmínka);
```

## Příklad použití "do-while" cyklu


Dokud není z klávesnice načtena požadovaná hodnota:




```
String vstup = "";
float cislo;
boolean nacteno; // vytvoř reader ze standardního vstupu
BufferedReader in = new BufferedReader(new InputStream(System.in));
// dokud není zadáno číslo, čti
do {
    vstup = in.readLine();
    try {
        cislo = Float.parseFloat(vstup);
        nacteno = true;
    } catch (NumberFormatException nfe) {
        nacteno = false;
    }
} while(!nacteno);
System.out.println("Nacteno cislo "+cislo);
```

Příklad:                      Načítej,                      dokud                      není                      zadáno                      číslo  
[\[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/DokudNeniZadano.java\]](http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/DokudNeniZadano.java)

## Cyklus "for"

obecnější než **for**  [\[http://www.instantweb.com/foldoc/foldoc.cgi?for\]](http://www.instantweb.com/foldoc/foldoc.cgi?for) v Pascalu, podobně jako v C/C++

De-facto jde o rozšíření **while**  [\[http://www.instantweb.com/foldoc/foldoc.cgi?while\]](http://www.instantweb.com/foldoc/foldoc.cgi?while), lze jím snadno nahradit

Syntaxe:

```
for(počáteční op.; vstupní podm.; příkaz po každém průch.)
    příkaz;
```

anebo (obvyklejší, bezpečnější)

```
for (počáteční op.; vstupní podm.; příkaz po každém průch.) {
    příkaz1;
    příkaz2;
    příkaz3;
    ...
}
```

## Příklad použití "for" cyklu

Provedení určité sekvence určitý počet krát

```
for (int i = 0; i < 10; i++) {
```

```
    System.out.println(i);  
}
```

Vypiše na obrazovku deset řádků s čísly postupně 0 až 9

1. Příklad: Pět pozdravů  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/PetPozdravu.java>]
2. Příklad: Výpis prvků pole objektů "for" cyklem  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/PolozkyForCyklem.java>]

## Doporučení k psaní `for` cyklů (1)

Používejte asymetrické intervaly (ostrá a neostrá nerovnost):

- podmínka daná počátečním přiřazením `i = 0` [G] [http://www.instantweb.com/foldoc/foldoc.cgi?i = 0] a inkrementací `i++` [G] [http://www.instantweb.com/foldoc/foldoc.cgi?i++] je *neostrou nerovností*, zatímco
- opakovací podmínka `i < 10` [G] [http://www.instantweb.com/foldoc/foldoc.cgi?i < 10] [G] [http://www.instantweb.com/foldoc/foldoc.cgi?] je *ostrou nerovností* -> `i` už nesmí hodnoty 10 dosáhnout!

Vytvarujte se složitých příkazů v hlavičce (kulatých závorkách) `for` [G] [http://www.instantweb.com/foldoc/foldoc.cgi?for] cyklu -

- je lepší to napsat podle situace před cyklus nebo až do jeho těla

## Doporučení k psaní `for` cyklů (2)

Někteří autoři nedoporučují psát deklaraci řídicí proměnné přímo do závorek cyklu

```
for (int i = 0; ... [G] [http://www.instantweb.com/foldoc/foldoc.cgi?for (int i = 0; ...]
```

ale rozepsat takto:

```
int i;
for (i = 0; ...
```

potom je proměnná `i` [http://www.instantweb.com/foldoc/foldoc.cgi?i] přístupná ("viditelná") i mimo cyklus - za cyklem, což se však ne vždy hodí.

## Vícecestné větvení "switch - case - default"

Obdoba pascalského **select - case - else** [http://www.instantweb.com/foldoc/foldoc.cgi?select - case - else]

Větvení do více možností na základě ordinální hodnoty

Syntaxe:

```
switch(výraz) {
    case hodnota1: prikaz1a;
                  prikaz1b;
                  prikaz1c;
                  ...
                  break;
    case hodnota2: prikaz2a;
                  prikaz2b;
                  ...
                  break;
    default:      prikazDa;
                  prikazDb;
                  ...
}
```

Je-li *výraz* roven některé z hodnot, provede se sekvence uvedená za příslušným **case** [http://www.instantweb.com/foldoc/foldoc.cgi?case].

Sekvenci obvykle ukončíme příkazem **break** [http://www.instantweb.com/foldoc/foldoc.cgi?break], který předá řízení ("skočí") na první příkaz za ukončovací závorkou příkazu **switch** [http://www.instantweb.com/foldoc/foldoc.cgi?switch].

Příklad: `Vícecestné větvení`  
[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/VicecestneVetveni.java]

## Vnořené větvení



Větvení **if - else** [http://www.instantweb.com/foldoc/foldoc.cgi?if - else] můžeme samozřejmě vnořovat do sebe:

```

if (podmínka_vnější) {
    if (podmínka_vnitřní_1) {
        ...
    } else {
        ...
    }
} else {
    if (podmínka_vnitřní_2) {
        ...
    } else {
        ...
    }
}

```

## Vnořené větvení (2)

Je možné "šetřit" a neuvádět složené závorky, v takovém případě se **else**  [\[http://www.instantweb.com/foldoc/foldoc.cgi?else\]](http://www.instantweb.com/foldoc/foldoc.cgi?else) vztahuje vždy k nejbližšímu neuzavřenému **if**  [\[http://www.instantweb.com/foldoc/foldoc.cgi?if\]](http://www.instantweb.com/foldoc/foldoc.cgi?if), např. znovu předchozí příklad:

```

if (podmínka_vnější)
    if (podmínka_vnitřní1)
        ...
    else // vztahuje se k nejbližšímu if
        // s if (podmínka_vnitřní_1)
        ...
else // vztahuje se k prvnímu if,
    // protože je v tuto chvíli
    // nejbližší neuzavřené
    if (podmínka_vnitřní_2)
        ...
    else // vztahuje se k if (podmínka_vnitřní_2)
        ...

```

Tak jako u cyklů - tento způsob zápisu nelze v žádném případě doporučit!!!

Příklad: Vnořené větvení  
[\[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/VnoreneVetveni.java\]](http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/VnoreneVetveni.java)

## Řetězené "if - else if - else"

Někdy rozvíjíme pouze druhou (negativní) větev:


```

if (podmínka1) {
    ...
} else if (podmínka2) {

```

```
...
} else if (podmínka3) {
    ...
} else {
    ...
}
```


Neplatí-li podmínka1, testuje se podmínka2, neplatí-li, pak podmínka3...


neplatí-li žádná, provede se příkaz za posledním - samostatným - **else**   
[<http://www.instantweb.com/foldoc/foldoc.cgi?else>].

Opět je dobré všude psát složené závorky!!!

Příklad: Řetězené if  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/VicecestneVetveniIf.java>]

## Příkazy "break"

Realizuje "násilné" ukončení průchodu cyklem nebo větvením **switch**   
[<http://www.instantweb.com/foldoc/foldoc.cgi?switch>]

Syntaxe použití **break**  [<http://www.instantweb.com/foldoc/foldoc.cgi?break>] v **cyklu**:

```
for (int i = 0; i < a.length; i++) {
    if(a[i] == 0) {
        break; // skoci se za konec cyklu
    }
}
if (a[i] == 0) {
    System.out.println("Nasli jsme 0 na pozici "+i);
} else {
    System.out.println("0 v poli neni");
}
```

použití u **switch**  [<http://www.instantweb.com/foldoc/foldoc.cgi?switch>] jsme již viděli, Vícecestné větvení "switch - case - default"









## Příkaz "continue"

Používá se v těle cyklu.

Způsobí přeskočení zbylé části průchodu tělem cyklu

```
for (int i = 0; i < a.length; i++) {
    if (a[i] == 5)
        continue;
```



```
System.out.println(i);
}
```

Výše uvedený příklad vypíše čísla 1  [http://www.instantweb.com/foldoc/foldoc.cgi?1], 2   
 [http://www.instantweb.com/foldoc/foldoc.cgi?2], 3   
 [http://www.instantweb.com/foldoc/foldoc.cgi?3], 4   
 [http://www.instantweb.com/foldoc/foldoc.cgi?4], 6   
 [http://www.instantweb.com/foldoc/foldoc.cgi?6], 7   
 [http://www.instantweb.com/foldoc/foldoc.cgi?7], 8   
 [http://www.instantweb.com/foldoc/foldoc.cgi?8], 9   
 [http://www.instantweb.com/foldoc/foldoc.cgi?9], nevypíše hodnotu 5   
 [http://www.instantweb.com/foldoc/foldoc.cgi?5].

Příklad: Řízení průchodu cyklem pomocí "break" a "continue"  
 [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/BreakContinue.java]

## "break" a "continue" s návěštím

Umožní ještě jemnější řízení průchodu vnořenými cykly:


- pomocí návěští můžeme naznačit, který cyklus má být příkazem **break**  [http://www.instantweb.com/foldoc/foldoc.cgi?break] přerušen nebo
- tělo kterého cyklu má být přeskočeno příkazem **continue**  [http://www.instantweb.com/foldoc/foldoc.cgi?continue].

Příklad: Návěští [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/rizeni/Navesti.java]

## Doporučení k příkazům break a continue





### Poznámka

Raději NEPOUŽÍVAT, ale jsou menším zlem než by bylo **goto**   
 [http://www.instantweb.com/foldoc/foldoc.cgi?goto] (kdyby v Javě existovalo...), protože nepředávají řízení dále než za konec struktury (cyklu, větvení).




### Poznámka

Toto však již neplatí pro **break**  [http://www.instantweb.com/foldoc/foldoc.cgi?break]  
 a **continue**  [http://www.instantweb.com/foldoc/foldoc.cgi?continue] na návěští!



## Poznámka

Poměrně často se používá **break**  [<http://www.instantweb.com/foldoc/foldoc.cgi?break>]  
při sekvenčním vyhledávání prvku.

---





# Kapitola 6. Ladění programů, testování jednotek (junit)

## Ladění programů (debugging)

- Ladění programů s debuggerem jdb
- Nástroje ověřování podmínek za běhu - klíčové slovo assert
- Nástroje testování jednotek (tříd, balíků) - junit
- Pokročilé systémy dynamického ověřování podmínek - jass

## Ladění programů v Javě

Je mnoho způsobů...

- kontrolní tisky - `System.err.println(...)`   
[[http://www.instantweb.com/foldoc/foldoc.cgi?System.err.println\(...\)](http://www.instantweb.com/foldoc/foldoc.cgi?System.err.println(...))]
- řádkovým debuggerem jdb  [<http://www.instantweb.com/foldoc/foldoc.cgi?jdb>]
- integrovaným debuggerem v IDE
- pomocí speciálních nástrojů na záznam běhu pg.:  
nejrůznější "logger" - standardní poskytuje od JDK1.4 balík `java.util.logging`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?java.util.logging>] nebo alternativní a zdařilejší `log4j`  [<http://www.instantweb.com/foldoc/foldoc.cgi?log4j>]

## Ještě lepší...

- je používat systémy pro běhovou kontrolu platnosti podmínek:
  - vstupní podmínka metody (zda je volána s přípustnými parametry)
  - výstupní podmínka metody (zda jsou dosažené výstupy správné)
  - a podmínka kdekoli jinde - např. invariant cyklu...
- K tomuto slouží jednak




- standardní klíčové slovo (od JDK1.4) `assert` booleovský výraz   
[<http://www.instantweb.com/foldoc/foldoc.cgi?assert> booleovský výraz ]
- testovací nástroje typu JUnit  [<http://www.instantweb.com/foldoc/foldoc.cgi?JUnit>] (a varianty - `HttpUnit`  [<http://www.instantweb.com/foldoc/foldoc.cgi?HttpUnit>],...) - s metodami `assertEquals()`  [<http://www.instantweb.com/foldoc/foldoc.cgi?assertEquals>] apod.
- pokročilé nástroje na běhovou kontrolu platnosti invariantů, vstupních, výstupních a dalších podmínek - např. `jass`  [<http://www.instantweb.com/foldoc/foldoc.cgi?jass>] (Java with ASSERTions), <http://csd.informatik.uni-oldenburg.de/~jass/>.
- Ukázka testu jednotky: `Test` třídy `ChovatelPsu`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?ChovatelPsu>]  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet/chovatelstvi/psi/ChovatelPsuTest.java>]

## Běhové ověření podmínky - `assert`

### Postup při práci s `assert`


[<http://www.instantweb.com/foldoc/foldoc.cgi?assert>]

Postup:

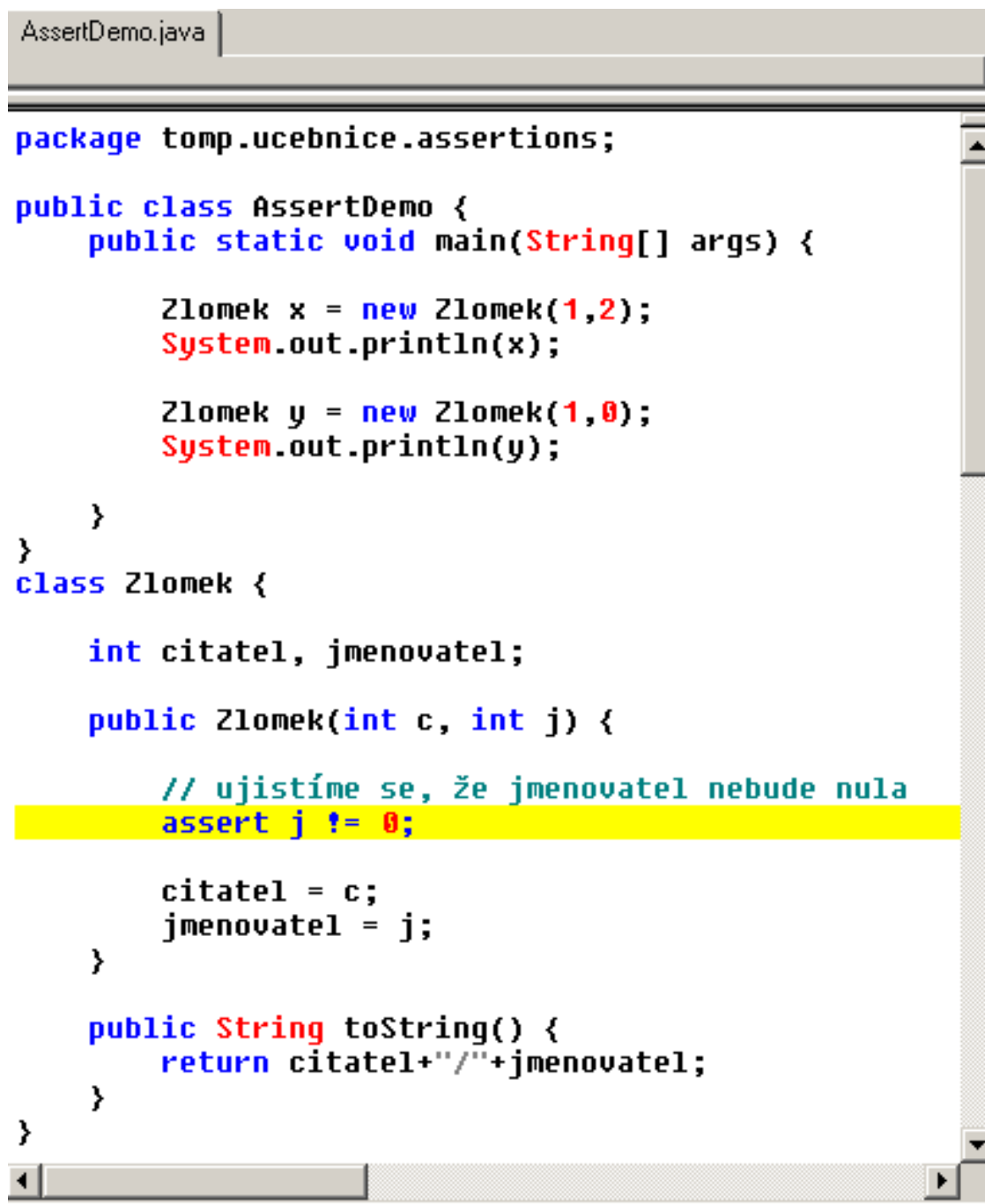
1. Napsat zdrojový program užívající klíčové slovo `assert`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?assert>] (pouze od verze Java2 v1.4 výše). Nepotřebujeme žádné speciální běhové knihovny, vše je součástí Javy; musíme ovšem mít překladové i běhové prostředí v1.4 a vyšší.
2. Přeložit jej s volbou `-source 1.4`
3. Spustit jej s volbou `-ea` (`-enableassertions`).  
  
Aktivovat aserce lze i selektivně pro některé třídy (`-ea název_třídy` nebo `-ea název_balíku...` - tři tečky na konci!!!).
4. Dojde-li za běhu programu k porušení podmínky stanovené za `assert`, vznikne běhová chyba (`AssertionError`) a program skončí.

### Ukázka použití `assert` (1)

Třída `Zlomek` používá `assert` k ověření, že zlomek není (chybou uživatele) vytvářen s nulovým jmenovatelem.

Za **assert**  [<http://www.instantweb.com/foldoc/foldoc.cgi?assert>] uvedeme, co musí v daném místě za běhu programu platit.

Obrázek 6.1. Třídy AssertDemo, Zlomek



```
AssertDemo.java

package tomp.ucebnice.assertions;

public class AssertDemo {
    public static void main(String[] args) {

        Zlomek x = new Zlomek(1,2);
        System.out.println(x);

        Zlomek y = new Zlomek(1,0);
        System.out.println(y);

    }
}

class Zlomek {

    int citatel, jmenovatel;

    public Zlomek(int c, int j) {

        // ujistíme se, že jmenovatel nebude nula
        assert j != 0;

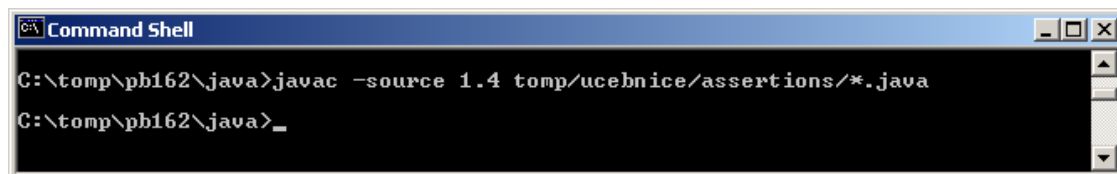
        citatel = c;
        jmenovatel = j;
    }

    public String toString() {
        return citatel+"/"+jmenovatel;
    }
}
```

## Ukázka použití assert (2)

Program přeložíme (s volbou `-source 1.4`):

Obrázek 6.2. Správný postup překladu AssertDemo

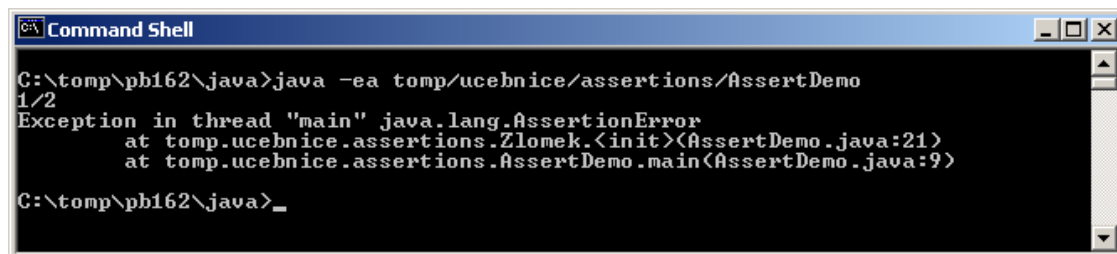


```
Command Shell
C:\tomp\pb162\java>javac -source 1.4 tomp/ucebnice/assertions/*.java
C:\tomp\pb162\java>_
```

## Ukázka použití assert (3)

Program spustíme (s volbou `-ea` nebo selektivním `-ea:NázevTřidy`):

Obrázek 6.3. Spuštění AssertDemo s povolením assert

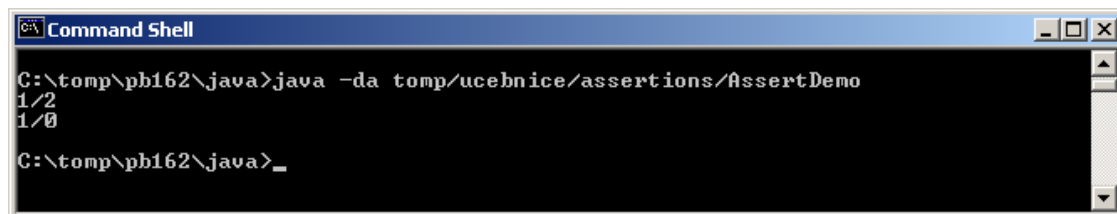


```
Command Shell
C:\tomp\pb162\java>java -ea tomp/ucebnice/assertions/AssertDemo
1/2
Exception in thread "main" java.lang.AssertionError
    at tomp.ucebnice.assertions.Zlomek.<init>(AssertDemo.java:21)
    at tomp.ucebnice.assertions.AssertDemo.main(AssertDemo.java:9)
C:\tomp\pb162\java>_
```

## Ukázka použití assert (4)

Spustíme-li bez povolení assert (bez volby `-ea` nebo naopak s explicitním zákazem: `-da`), pak program podmínky za assert neověřuje:

Obrázek 6.4. Spuštění AssertDemo bez povolení assert



```
Command Shell
C:\tomp\pb162\java>java -da tomp/ucebnice/assertions/AssertDemo
1/2
1/0
C:\tomp\pb162\java>_
```

## Testování jednotek nástrojem junit

## Postup při práci s JUnit

### [<http://www.instantweb.com/foldoc/foldoc.cgi?JUnit>]

Uvědomit si, že žádný nástroj za nás nevymyslí, JAK máme své třídy testovat. Pouze nám napomůže ke snadnějšímu sestavení a spuštění testu.

Postup:

1. Stáhnout si z <http://junit.org> poslední (stačí binární) distribuci testovacího prostředí.
2. Nainstalovat JUnit  [<http://www.instantweb.com/foldoc/foldoc.cgi?JUnit>] (stačí rozbalit do samostatného adresáře).
3. Napsat testovací třídu/třídy - buďto implementují rozhraní `junit.framework.Test`  [<http://www.instantweb.com/foldoc/foldoc.cgi?junit.framework.Test>] nebo obvykleji rovnou rozšiřují třídu `junit.framework.TestCase`  [<http://www.instantweb.com/foldoc/foldoc.cgi?junit.framework.TestCase>]
4. Testovací třída obsahuje metodu na nastavení testu (`setUp`), testovací metody (`testNeco`) a úklidovou metodu (`tearDown`).
5. Testovací třídu spustit v prostředí (řádkovém nebo GUI) - `junit.textui.TestRunner`  [<http://www.instantweb.com/foldoc/foldoc.cgi?junit.textui.TestRunner>],  
`junit.swingui.TestRunner`  [<http://www.instantweb.com/foldoc/foldoc.cgi?junit.swingui.TestRunner>]...
6. Testovač zobrazí, které testovací metody případně selhaly.

## Ukázka použití JUnit (1)

Třída `Zlomek` zůstává zhruba jako v předchozím příkladu, přibývají však metody `equals` (porovnává dva zlomky, zda je jejich číselná hodnota stejná) a `soucet` (sečítá dva zlomky, součet vrací jako výsledek).

**Obrázek 6.5. Upravená třída `Zlomek` s porovnáním a součtem**

JUnitDemo.java | Assert.java |

```
}  
class Zlomek {  
    int citatel, jmenovatel;  
    public Zlomek(int c, int j) {  
        assert j != 0;  
        citatel = c;  
        jmenovatel = j;  
    }  
    public boolean equals(Object o) {  
        if (o instanceof Zlomek) {  
            Zlomek z = (Zlomek)o;  
            return citatel * z.jmenovatel == z.citatel * jmenovatel;  
        } else  
            return false;  
    }  
    public static Zlomek soucet(Zlomek a, Zlomek b) {  
        return new Zlomek(a.citatel*b.jmenovatel + b.citatel*a.jmenovatel,  
                           a.jmenovatel*b.jmenovatel);  
    }  
    public String toString() {  
        return citatel+"/"+jmenovatel;  
    }  
}
```

## Ukázka použití JUnit (2)

Testovací třída JUnitDemo má „přípravnou“ metodu setUp, tearDown a testovací metody.

**Obrázek 6.6.** Testovací třída JUnitDemo

```

JUnitDemo.java | Assert.java |
package tomp.ucebnice.junit;

public class JUnitDemo extends junit.framework.TestCase {

    private Zlomek x, y, soucet, xx;

    public void setUp() {
        x = new Zlomek(1,2);
        xx = new Zlomek(6, 12);
        y = new Zlomek(1,3);
        soucet = new Zlomek(5,6);
    }

    public void testRovna() {
        assertEquals("1/2 a 6/12 se musi rovnat", x, xx);
    }

    public void testSoucetRovna() {
        Zlomek s = Zlomek.soucet(x, y);
        assertEquals("Soucet 1/2 a 1/3 se musi rovnat 5/6", soucet, s);
    }

    public void testNelzeVytvorit() {
        try {
            Zlomek bad = new Zlomek(10,0);
            fail("Zlomek s nulovym jmenovatelem nemel jit vytvorit");
        } catch (AssertionError ae) {
            // OK!
        }
    }

    public void tearDown() {
    }

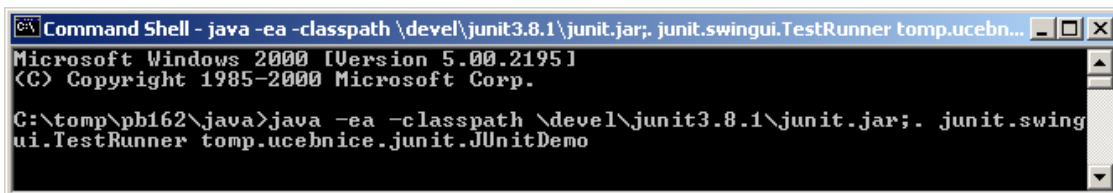
}

```

## Ukázka použití JUnit (3)

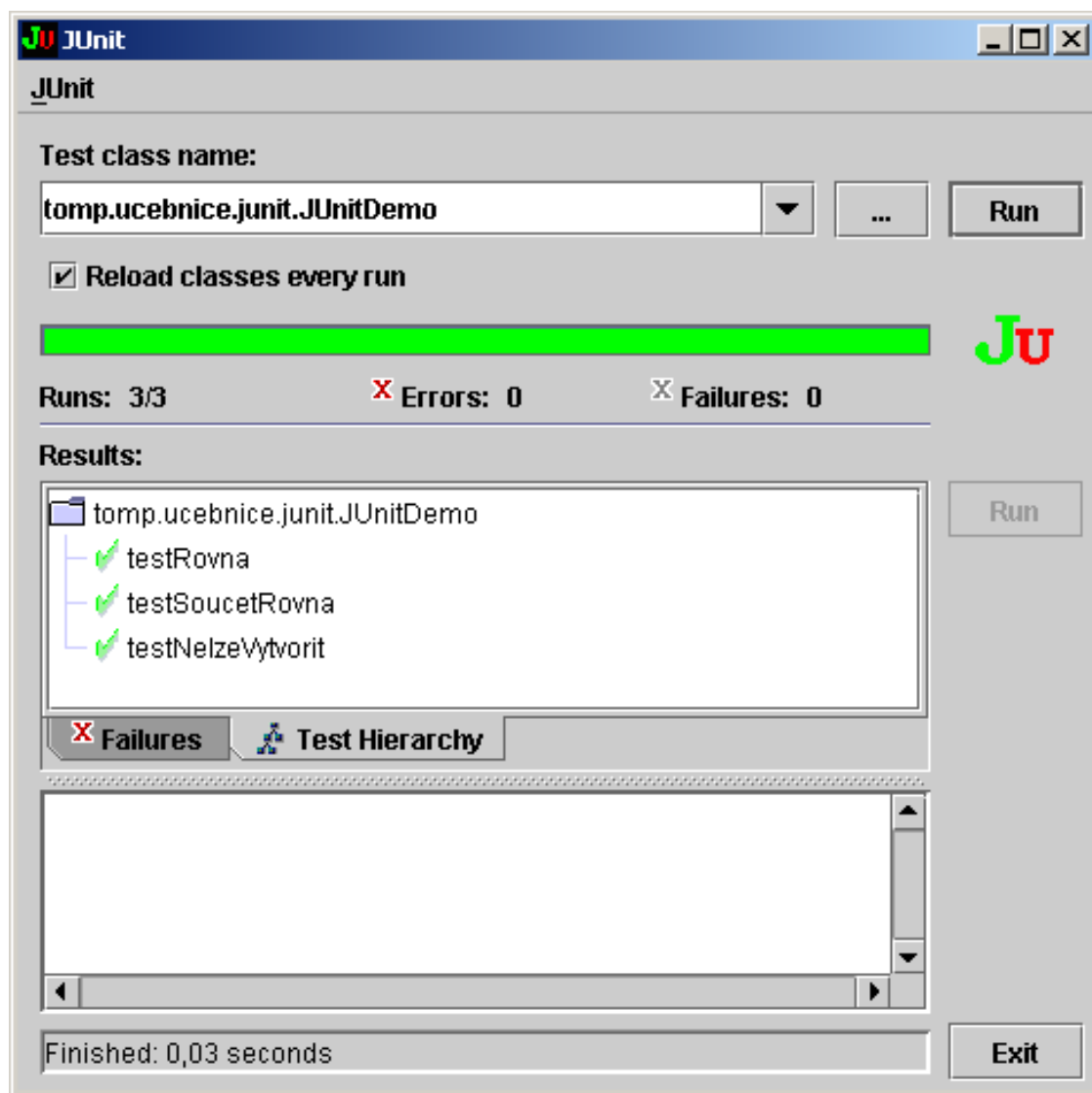
Spuštění testovače v prostředí GUI Swing nad testovací třídou JUnitDemo.

Obrázek 6.7. Spouštění testovače nad testovací třídou JUnitDemo



Pokud testovací třída prověří, že testovaná třída/y je/jsou OK, vypadá to přibližně takto:

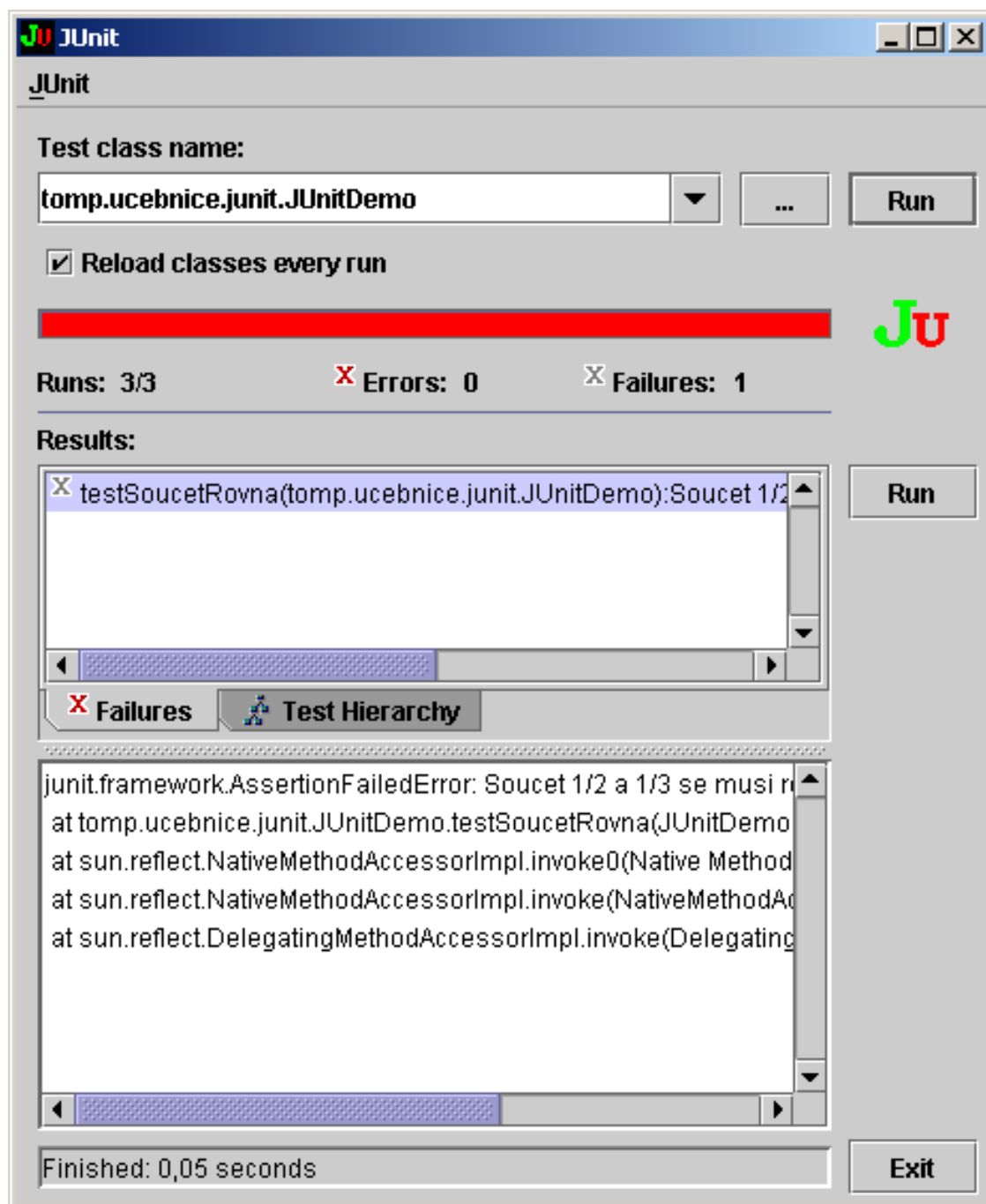
Obrázek 6.8. Testovač spouštějící třídu JUnitDemo



## Ukázka použití JUnit (4)

Má-li testovaná třída/y chyby a zjistí-li to testovač, vypadá to třeba takto:

Obrázek 6.9. Testovací třída JUnitDemo našla chybu



## Návrh dle kontraktu

Postup při práci s jass   
[\[http://www.instantweb.com/foldoc/foldoc.cgi?jass\]](http://www.instantweb.com/foldoc/foldoc.cgi?jass)

jass je preprocesor javového zdrojového textu. Umožňuje ve zdrojovém textu programu vyznačit pod-










mínky, jejichž splnění je za běhu kontrolováno.

Podmínkami se rozumí:

- pre- a postconditions u metod (vstupní a výstupní podmínky metod)
- invarianty objektů - podmínky, které zůstávají pro objekt v platnosti mezi jednotlivými operacemi nad objektem

Postup práce s jass:

- stažení a instalace balíku z <http://csd.informatik.uni-oldenburg.de/~jass/>
- vytvoření zdrojového textu s příponou `.jass`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?.jass>], javovou syntaxí s použitím speciálních komentářových značek
- takový zdrojový text je přeložitelný i normálním překladačem `javac`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?javac>], ale v takovém případě ztrácíme možnosti jass
- proto nejprve `.jass`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.jass>] souboru převedeme preprocesorem jass na javový (`.java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.java>]) soubor
- ten již přeložíme `javac`  [<http://www.instantweb.com/foldoc/foldoc.cgi?javac>] a spustíme `java`  [<http://www.instantweb.com/foldoc/foldoc.cgi?java>], tedy jako každý jiný zdrojový soubor v Javě
- z `.jass`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.jass>] zdrojů je možné vytvořit také dokumentaci API obsahující jass značky, tj. informace, co kde musí platit za podmínky atd. - vynikající možnost!

## Odkazy

- JUnit homepage [<http://junit.org>]
- Java 1.4 logging API guide [<http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/>]
- Log4j homepage [<http://jakarta.apache.org/log4j/docs/index.html>]
- jass homepage [<http://csd.informatik.uni-oldenburg.de/~jass/>]
- úvodní materiálek [<http://www.inf.fu-berlin.de/lehre/SS01/VIS/Dokumente/Vortraege/junit.pdf>] k použití junit (v němčině, jako PDF)



---

# Kapitola 7. Pokročilejší objektový návrh. dědičnost rozhraní, implementace více rozhraní. Abstraktní třídy.

## Objektové modelování v Javě - pokračování

- Implementace více rozhraní jednou třídou
- Rozšiřování rozhraní (dědičnost mezi rozhraními)
- Rozšiřování více rozhraní (vícenásobná dědičnost mezi rozhraními)
- Abstraktní třídy (částečná implementace)

## Implementace více rozhraní současně

### Implementace více rozhraní současně

Třída sice smí dědit maximálně z jedné nadtřídy (předka), ale

- zato může současně implementovat libovolný počet rozhraní!
- Podmínkou ovšem je, aby se metody ze všech implementovaných rozhraní „snesly“ v jedné třídě.
- Které že se nesnesou? Např. dvě metody se skoro stejnou hlavičkou, lišící se „jen“ návratovým typem...

### Implementace více rozhraní současně - příklad

Příklad - kromě výše uvedeného intf. *Informujici* mějme ještě:

```
public interface Kricici {  
    void zakric();  
}
```

Třída Clovek implementuje dvě rozhraní:

```
public class Clovek
```

```
implements Informujici, Kricici {  
    ...  
    public void vypisInfo() {  
        ...  
    }  
    public void zakric() {  
        ...  
    }  
}
```

## Rozšiřování rozhraní

### Rozšiřování rozhraní

Podobně jako u tříd, i rozhraní mohou být rozšiřována/specializována. Mohou dědit.

Na rozdíl od třídy, která dědí maximálně z jedné nadtřídy (předka) -

- z rozhraní můžeme odvozovat potomky (podrozhraní - *subinterfaces*)
- dokonce i *vícenásobně* - z více rozhraní odvodíme společného potomka slučujícího a rozšiřujícího vlastnosti všech předků.

Přesto to nepřináší problémy jako u klasické plné vícenásobné dědičnosti např. v C++, protože rozhraní samo

- nemá proměnné
- metody neimplementuje
- nedochází tedy k nejednoznačnostem a konfliktům při poddědění neprázdných, implementovaných metod a proměnných

### Rozšiřování rozhraní - příklad

Příklad - *Informujici* informuje „jen trochu“, *DobreInformujici* je schopen ke standardním informacím (*vypisInfo*) přidat dodatečné informace (*vypisDodatecneInfo*).


```
public interface Informujici {  
    void vypisInfo();  
}  
  
public interface DobreInformujici extends Informujici {  
    void vypisDodatecneInfo();  
}
```

```
}
```

Třída, která chce implementovat `intf.DobreInformujici`, musí implementovat *obě* metody předepsané tímto rozhraním. Např.:

```
public class Informator implements DobreInformujici {  
    public void vypisInfo() {  
        ... // kód metody  
    }  
    public void vypisDodatecneInfo() {  
        ... // kód metody  
    }  
}
```

## Rozhraní - poznámky

- Používají se i prázdná rozhraní - nepředepisující žádnou metodu
- deklarace, že třída implementuje také rozhraní, ji "k ničemu nezavazuje", ale poskytuje typovou informaci o dané třídě
- i v Java Core API jsou taková rozhraní - např. `java.lang.Cloneable`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?java.lang.Cloneable>]

## Abstraktní třídy


### Abstraktní třídy

I když Java disponuje rozhraními, někdy je vhodné určitou specifikaci implementovat pouze *částečně*:

- Rozhraní = Specifikace
- Abstraktní třída = Částečná implementace
- Třída = Implementace

### Abstraktní třídy (2)

Abstraktní třída je tak označena v hlavičce, např.:

```
public          abstract          class          AbstraktniChovatel          ...   
[http://www.instantweb.com/foldoc/foldoc.cgi?public abstract class AbstraktniChovatel ...]
```

Obvykle má alespoň jednu *abstraktní metodu*, deklarovanou např.:

```
public          abstract          void          vypisInfo()          ...
```

[[http://www.instantweb.com/foldoc/foldoc.cgi?public abstract void vypisInfo\(\) ...](http://www.instantweb.com/foldoc/foldoc.cgi?public+abstract+void+vypisInfo()+...)]

Od a.t. *nelze vytvořit instanci*, nelze napsat např.:

```
Chovatel          ch          =          new          AbstraktniChovatel (...);
```

[[http://www.instantweb.com/foldoc/foldoc.cgi?Chovatel ch = new AbstraktniChovatel\(...\);](http://www.instantweb.com/foldoc/foldoc.cgi?Chovatel+ch+=new+AbstraktniChovatel(...);)]

## Příklad rozhraní - abstraktní třída - neabstraktní třída

Viz Svět chovatelství [<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet.html>] z učebnice:

- rozhraní `svet.chovatelstvi.Chovatel`  
[<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.Chovatel>] - specifikace, co má chovatel umět
- `svet.chovatelstvi.AbstraktniChovatel`  
[<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.AbstraktniChovatel>] - částečná implementace chovatele
- `svet.chovatelstvi.psi.ChovatelPsu`  
[<http://www.instantweb.com/foldoc/foldoc.cgi?svet.chovatelstvi.psi.ChovatelPsu>] - úplná implementace chovatele psů

Pozn.: Obecný chovatel se ihned úplně implementovat nedá (ještě to neumíme), proto je definován jako *abstraktní třída* *AbstraktniChovatel* a teprve až *ChovatelPsu* je *neabstraktní třída*.

---

# Kapitola 8. Dynamické datové struktury (kontejnery).

## Kontejnery, iterátory, kolekce


- Kontejnery jako základní dynamické struktury v Javě
- Kolekce, iterátory (Collection, Iterator)
- Seznamy (rozhraní List, třídy ArrayList, LinkedList)
- Množiny (rozhraní Set, třída HashSet), uspořádané množiny (rozhraní SortedSet, třída TreeSet), rozhraní Comparable, Comparator
- Mapy (rozhraní Map, třída HashMap), uspořádané mapy (rozhraní SortedMap, třída TreeMap)
- Klasické netypové vs. nové typové kontejnery - generické datové typy
- Iterace cyklem foreach
- Starší typy kontejnerů (Vector, Stack, Hashtable)

## Kontejnery

*Kontejnery* (containers) v Javě

- slouží k ukládání objektů (ne hodnot primitivních typů!)
- v Javě koncipovány dosud jako *beztypové* - to se ve verzi 1.5 částečně změní!
- tím se liší od např. *Standard Template Library* v C++

Většinou se používají kontejnery hotové, vestavěné, tj. ty, jež jsou součástí Java Core API:

- vestavěné kontejnerové třídy jsou definovány v balíku `java.util`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?java.util>]
- je možné vytvořit si vlastní implementace, obvykle ale zachovávající/implementující „standardní“ rozhraní

K čemu slouží?

- jsou dynamickými alternativami k poli a mají daleko širší použití
- k uchování proměnného počtu objektů -
- počet prvků se v průběhu existence kontejneru může měnit
- oproti polím nabízejí časově efektivnější algoritmy přístupu k prvkům

## Základní kategorie kontejnerů

- seznam (*List*) - lineární struktura
- množina (*Set*) - struktura bez uspořádání, rychlé dotazování na přítomnost prvku
- asociativní pole, mapa (*Map*) - struktura uchovávající dvojice klíč->hodnota, rychlý přístup přes klíč

## Kontejnery - rozhraní, nepovinné metody

- Funkcionalita vestavěných kontejnerů je obvykle předepsána výhradně *rozhraním*, jenž implementují.
- Rozhraní však připouští, že některé metody jsou *nepovinné*, třídy jej nemusí implementovat!
- V praxi se totiž někdy nehodí implementovat jak čtecí, tak i zápisové operace - některé kontejnery jsou „read-only“

## Kontejnery - souběžný přístup, výjimky

- Moderní kontejnery jsou *nesynchronizované*, nepřipouštějí souběžný přístup z více vláken.
- Standardní, nesynchronizovaný, kontejner lze však „zabalit“ synchronizovanou obálkou.
- Při práci s kontejnery může vzniknout řada *výjimek*, např. *IllegalStateException* apod.
- Většina má charakter výjimek *běhových*, není povinností je odchytávat - pokud věříme, že nevzniknou.

## Iterátory

*Iterátory* jsou prostředkem, jak "chodit" po prvcích kolekce buďto

- v neurčeném pořadí nebo
- v uspořádání (u uspořádaných kolekcí)



Každý iterátor musí implementovat velmi jednoduché rozhraní *Iterator* se třemi metodami:

- *boolean hasNext()*
- *Object next()*
- *void remove()*

## Kolekce

- jsou kontejnery implementující rozhraní *Collection* - API doc k rozhr. *Collection* [<http://java.sun.com/j2se/1.4/docs/api/java/util/Collection.html>]
- Rozhraní kolekce popisuje velmi obecný kontejner, disponující operacemi: *přidávání*, *rušení prvku*, *získání iterátoru*, *zjišťování prázdnosti* atd.
- Mezi kolekce patří mimo *Mapy* všechny ostatní vestavěné kontejnery - *List*, *Set*
- Prvky kolekce nemusí mít svou pozici danou indexem - viz např. *Set*

## Seznamy

### Seznamy

- lineární struktury
- implementují rozhraní *List* (rozšíření *Collection*) API doc k rozhr. *List* [<http://java.sun.com/j2se/1.4/docs/api/java/util/List.html>]
- prvky lze adresovat indexem (typu *int*)
- poskytují možnost získat dopředný i zpětný *iterátor*
- lze pracovat i s *podseznamy*

## Seznamy a obyčejné iterátory - příklad

Vytvoříme seznam, naplníme jej a chodíme po položkách iterátorem.

### Obrázek 8.1. Pohyb po seznamu iterátorem

```

IteratorDemo.java

package tomp.ucebnice.kolekce;

import java.util.*;

public class IteratorDemo {
    public static void main(String[] args) {

        // vytvoříme prázdný seznam
        List seznam = new ArrayList();

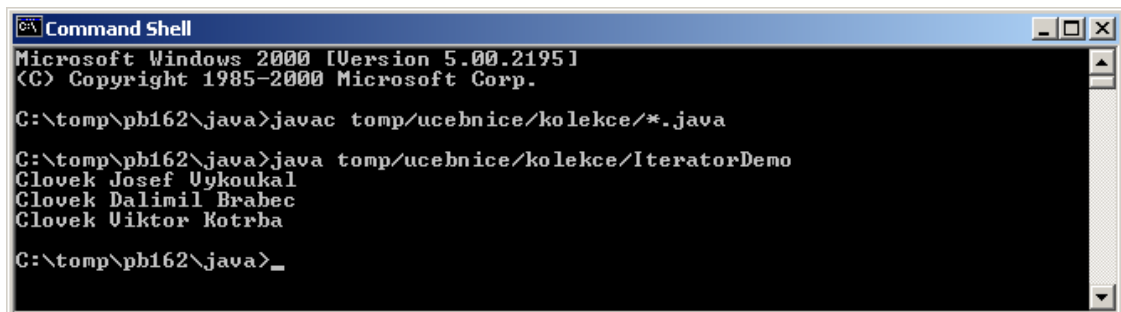
        // naplníme jej třemi lidmi
        seznam.add(new Clovek("Josef", "Uykoukal"));
        seznam.add(new Clovek("Dalimil", "Brabec"));
        seznam.add(new Clovek("Viktor", "Kotrba"));

        // projdeme iterátorem a vypíšeme postupně všechny lidi
        for(Iterator i = seznam.iterator(); i.hasNext(); ) {
            // získěj další objekt: je to člověk, můžeš přetypovat
            Clovek c = (Clovek)i.next();
            c.vypisInfo();
        }

        // Clovek bude vnořená třída, aby nebyla vůbec vidět ven -
        // - jiný význam to nemá...
        static class Clovek {
            String jmeno, prijmeni;
            Clovek (String j, String p) {
                jmeno = j;
                prijmeni = p;
            }
            public void vypisInfo() {
                System.out.println("Clovek "+jmeno+" "+prijmeni);
            }
        }
    }
}

```

Obrázek 8.2. Spuštění pg. s iterátorem



```

Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java

C:\tomp\pb162\java>java tomp/ucebnice/kolekce/IteratorDemo
Clovek Josef Uykoukal
Clovek Dalimil Brabec
Clovek Viktor Kotrba

C:\tomp\pb162\java>_

```

## Iterátor po seznamu - příklad

Vytvoříme seznam, naplníme jej a chodíme po položkách seznamovým iterátorem, vytvořeným na určité pozici (indexu) v seznamu.

Obrázek 8.3. Pohyb seznamovým iterátorem

ListIteratorDemo.java

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

    // vytvoříme prázdný seznam
    List seznam = new ArrayList();

    // naplníme jej třemi lidmi
    seznam.add(new Clovek("Josef", "Uykoukal"));
    seznam.add(new Clovek("Dalimil", "Brabec"));
    seznam.add(new Clovek("Viktor", "Kotrba"));

    // získáme seznamový iterátor od pozice 1
    // - umí víc než obyčejný!
    ListIterator li = seznam.listIterator(1);

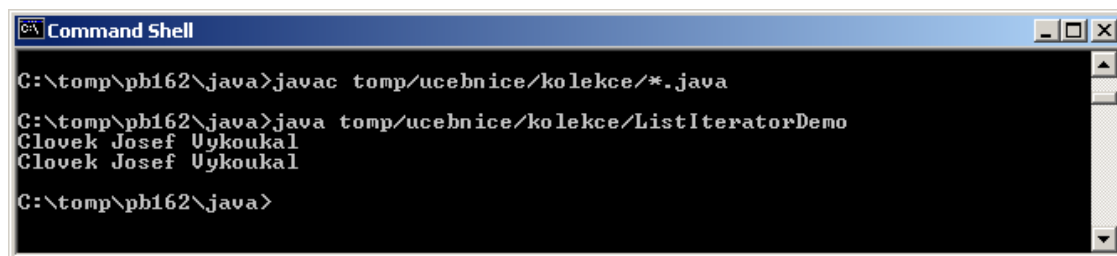
    // posuň se pozpátku na pozici 0
    // a získej objekt z pozice 0
    Clovek c = (Clovek)li.previous();
    c.vypisInfo();

    // získej aktuální objekt z pozice 0
    // a posuň se na 1
    c = (Clovek)li.next();
    c.vypisInfo();

}
```

K procházení seznamovým iterátorem lze použít metody *next*, *previous*.

Obrázek 8.4. Spuštění pg. se seznamovým iterátorem



```
Command Shell
C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/ListIteratorDemo
Clovek Josef Uykoukal
Clovek Josef Uykoukal
C:\tomp\pb162\java>
```

## Množiny

### Množiny

#### *Množiny*

- jsou struktury standardně bez uspořádání prvků (ale existují i uspořádané, viz dále)
- implementují rozhraní *Set* (což je rozšíření *Collection*)

Cílem množin je mít možnost rychle (se složitostí  $O(\log(n))$ ) provádět atomické operace:

- vkládání prvku (*add*)
- odebrání prvku (*remove*)
- dotaz na přítomnost prvku (*contains*)
- lze testovat i relaci „je podmnožinou“

Standardní implementace množiny:

- *hašovací tabulka* (*HashSet*) nebo
- *vyhledávací strom* (černobílý strom, Red-Black Tree - *TreeSet*)

## Množina - příklad

Vložíme prvky do množiny a ptáme se, zda tam jsou:

### Obrázek 8.5. Vložení prvků do množiny a dotaz na přítomnost

```

SetDemo.java

package tomp.ucebnice.kolekce;

import java.util.*;

public class SetDemo {
    public static void main(String[] args) {

        // vytvoříme prázdnou množinu
        Set mnozina = new HashSet();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Uykoukal");
        mnozina.add(c1);

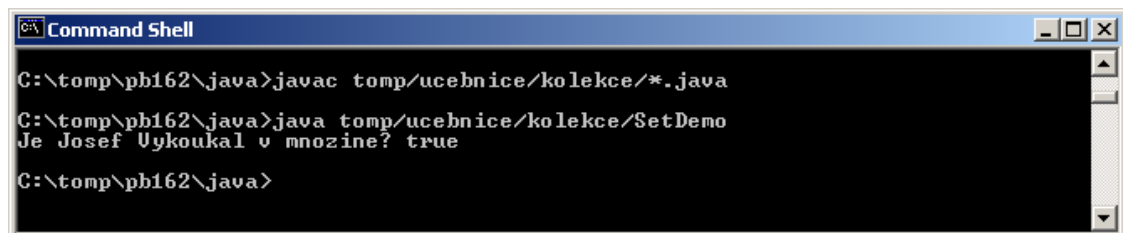
        Clovek c2 = new Clovek("Dalimil", "Brabec");
        mnozina.add(c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        mnozina.add(c3);

        System.out.println("Je Josef Uykoukal v mnozine? "
                           +mnozina.contains(c1));
    }
}

```

Obrázek 8.6. Spuštění pg. s množinou



```

C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/SetDemo
Je Josef Uykoukal v mnozine? true
C:\tomp\pb162\java>

```

## Uspořádané množiny

*Uspořádané množiny:*

- Implementují rozhraní *SortedSet* -API doc k rozhraní *SortedSet* [<http://java.sun.com/j2se/1.4/docs/api/java/util/SortedSet.html>]
- Jednotlivé prvky lze tedy iterátorem procházet v přesně definovaném pořadí - uspořádání podle *hodnot prvků*.

- Existuje vestavěná impl. *TreeSet* - černobílé stromy (Red-Black Trees) API doc ke třídě *TreeSet* [<http://java.sun.com/j2se/1.4/docs/api/java/util/TreeSet.html>]

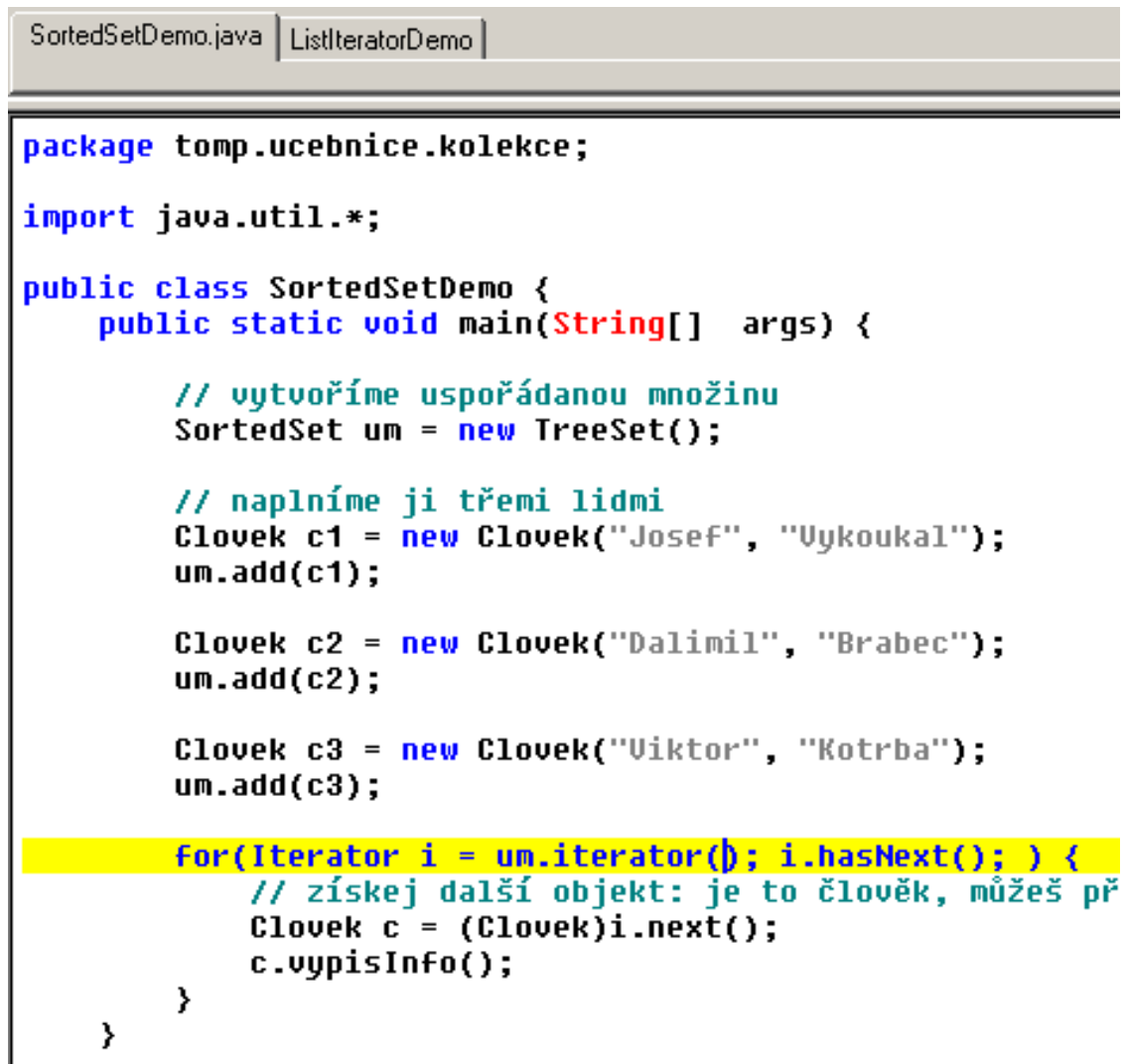
Uspořádání je dáno buďto:

- standardním chováním metody *compareTo* vkládaných objektů - pokud implementují rozhraní *Comparable*
- nebo je možné uspořádání definovat pomocí tzv. *komparátoru* (objektu impl. rozhraní *Comparator*) poskytnutých při vytvoření množiny.

## Uspořádaná množina - příklad s chybou

Vložíme prvky do uspořádané množiny. Prvky musejí implementovat rozhraní *Comparable*, nebo musíme poskytnout komparátor. Když neuděláme ani jedno:

**Obrázek 8.7. Vložení neporovnatelných prvků do uspořádané množiny**



```

SortedSetDemo.java | ListIteratorDemo

package tomp.ucebnice.kolekce;

import java.util.*;

public class SortedSetDemo {
    public static void main(String[] args) {

        // vytvoříme uspořádanou množinu
        SortedSet um = new TreeSet();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Uykoukal");
        um.add(c1);

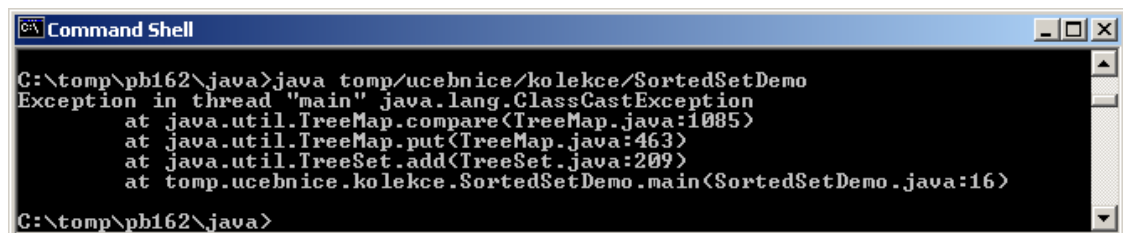
        Clovek c2 = new Clovek("Dalimil", "Brabec");
        um.add(c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        um.add(c3);

        for(Iterator i = um.iterator(); i.hasNext(); ) {
            // získěj další objekt: je to člověk, můžeš př
            Clovek c = (Clovek)i.next();
            c.vypisInfo();
        }
    }
}

```

Obrázek 8.8. Spuštění nefunkčního pg. s uspořádanou množinou



```

Command Shell

C:\tomp\pb162\java>java tomp/ucebnice/kolekce/SortedSetDemo
Exception in thread "main" java.lang.ClassCastException
    at java.util.TreeMap.compare(TreeMap.java:1085)
    at java.util.TreeMap.put(TreeMap.java:463)
    at java.util.TreeSet.add(TreeSet.java:209)
    at tomp.ucebnice.kolekce.SortedSetDemo.main(SortedSetDemo.java:16)

C:\tomp\pb162\java>

```

Nefunguje, prvky třídy *Clovek* nebyly porovnatelné.

## Uspořádaná množina - příklad OK

Prvky implementují rozhraní Comparable:

Obrázek 8.9. Vložení porovnatelných prvků do uspořádané množiny



```

SortedSetDemoOK.java | ListIteratorDemo
package tomp.ucebnice.kolekce;

import java.util.*;

public class SortedSetDemoOK {
    public static void main(String[] args) {

        // vytvoříme uspořádanou množinu
        SortedSet um = new TreeSet();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Uykoukal");
        um.add(c1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        um.add(c2);

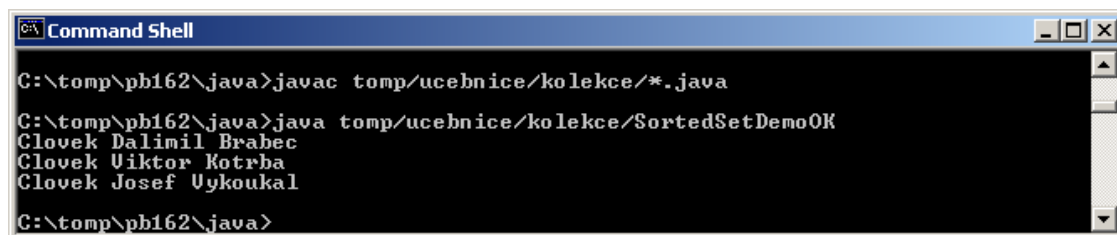
        Clovek c3 = new Clovek("Viktor", "Kotrba");
        um.add(c3);

        for(Iterator i = um.iterator(); i.hasNext(); ) {
            // získěj další objekt: je to člověk, můžeš přetypovat
            Clovek c = (Clovek)i.next();
            c.vypisInfo();
        }
        static class Clovek implements Comparable {
            String jmeno, prijmeni;
            Clovek (String j, String p) {
                jmeno = j;
                prijmeni = p;
            }
            public void vypisInfo() {
                System.out.println("Clovek "+jmeno+" "+prijmeni);
            }
            public int compareTo(Object o) {
                if (o instanceof Clovek) {
                    Clovek c = (Clovek)o;
                    return prijmeni.compareTo(c.prijmeni);
                } else
                    throw new IllegalArgumentException(
                        "Nelze porovnat objekt typu Clovek s objektem jineho typu")
            }
        }
    }
}

```

Obrázek 8.10. Spuštění funkčního pg. s uspořádanou množinou





```
C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/SortedSetDemoOK
Clovek Dalimil Brabec
Clovek Viktor Kotrba
Clovek Josef Uykoukal
C:\tomp\pb162\java>
```

Funguje, prvky třídy *Clovek* jsou porovnatelné, množina je uspořádána podle příjmení lidí.

## Mapy

### Mapy

*Mapy* (asociativní pole, nepřesně také hašovací tabulky nebo haše) fungují v podstatě na stejných principech a požadavcích jako *Set*:

- Ukládají ovšem dvojice (klíč, hodnota) a umožňují rychlé vyhledání dvojice podle hodnoty klíče.
- Základními metodami jsou: dotazy na přítomnost klíče v mapě (*containsKey*),
- výběr hodnoty odpovídající zadanému klíči (*get*),
- možnost získat zvlášť *množiny klíčů*, *hodnot* nebo *dvojic* (klíč, hodnota).

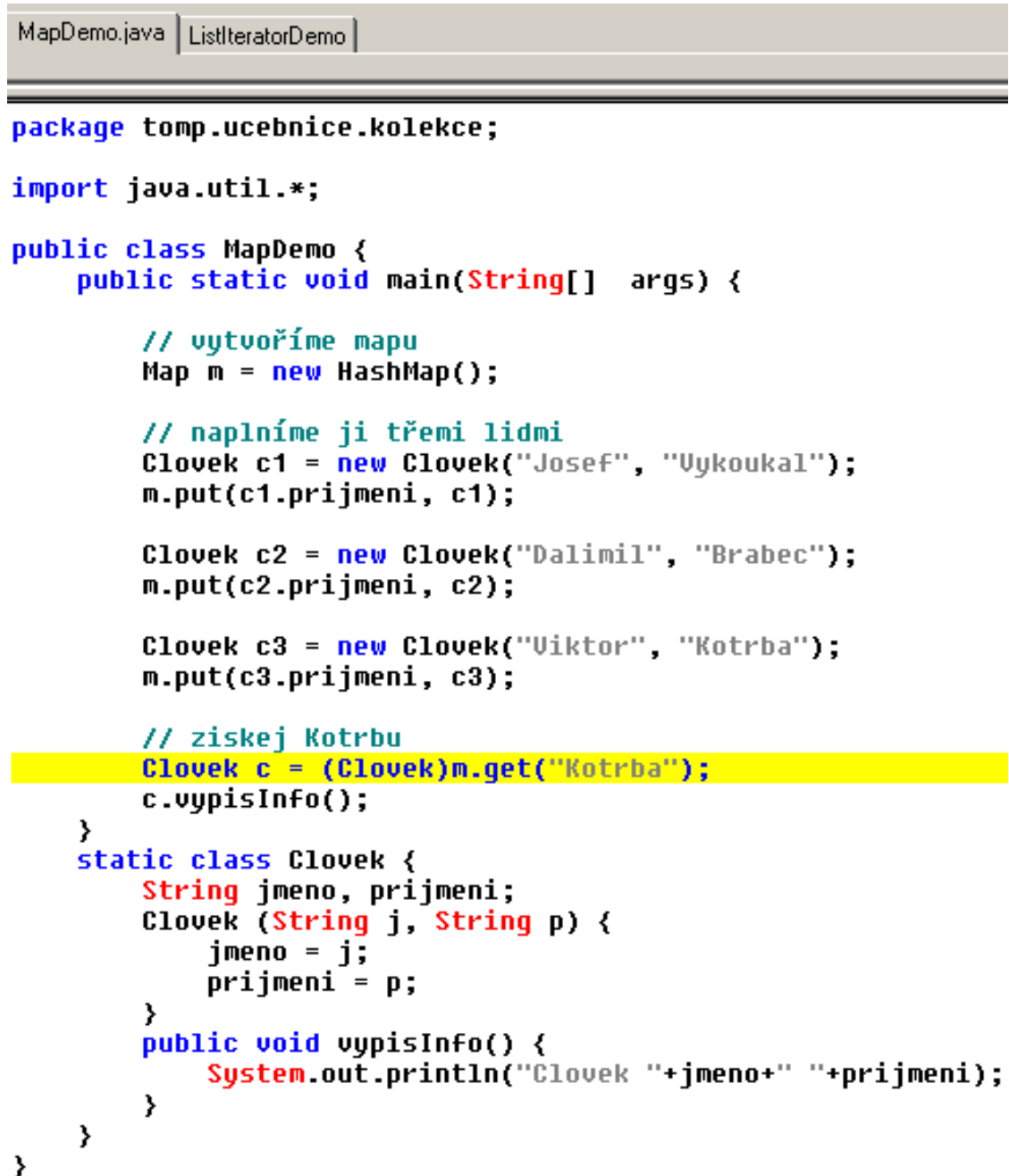
Mapy mají:

- podobné implementace jako množiny (tj. hašovací tabulky nebo stromy).
- logaritmickou složitost základních operací (*put*, *remove*, *containsKey*)

### Mapa - příklad

Lidi se do mapy vkládají s klíčem = příjmení člověka, pak se přes příjmení mohou vyhledat:

**Obrázek 8.11. Vložení lidí do mapy pod klíčem příjmení a vyhledání člověka**



```

package tomp.ucebnice.kolekce;

import java.util.*;

public class MapDemo {
    public static void main(String[] args) {

        // vytvoříme mapu
        Map m = new HashMap();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        m.put(c1.prijmeni, c1);

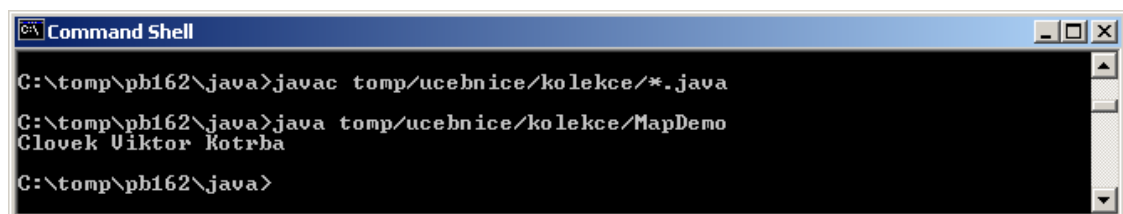
        Clovek c2 = new Clovek("Dalimil", "Brabec");
        m.put(c2.prijmeni, c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        m.put(c3.prijmeni, c3);

        // ziskej Kotrba
        Clovek c = (Clovek)m.get("Kotrba");
        c.vypisInfo();
    }
    static class Clovek {
        String jmeno, prijmeni;
        Clovek (String j, String p) {
            jmeno = j;
            prijmeni = p;
        }
        public void vypisInfo() {
            System.out.println("Clovek "+jmeno+" "+prijmeni);
        }
    }
}

```

Obrázek 8.12. Spuštění funkčního pg. s mapou



```

C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/MapDemo
Clovek Viktor Kotrba
C:\tomp\pb162\java>

```

## Uspořádané mapy

*Uspořádané mapy:*

- Implementují rozhraní *SortedMap* - API doc k rozhraní *SortedMap* [<http://java.sun.com/j2se/1.4/docs/api/java/util/SortedMap.html>]
- Dvojice (klíč, hodnota) jsou v nich *uspořádané podle hodnot klíče*.
- Existuje vestavěná impl. *TreeMap* - černobílé stromy (Red-Black Trees) - API doc ke třídě *TreeMap* [<http://java.sun.com/j2se/1.4/docs/api/java/util/TreeMap.html>]
- Uspořádání lze ovlivnit naprosto stejným postupem jako u uspořádané množiny.

## Uspořádaná mapa - příklad

Jsou-li klíče uspořádané (pomocí implementace *Comparable* nebo komparátorem), mohou se prvky procházet podle uspořádání klíčů:

**Obrázek 8.13. Vložení lidí do mapy pod uspořádaným klíčem příjmení a projde je**

```

SortedMapDemo.java | ListIteratorDemo |
package tomp.ucebnice.kolekce;

import java.util.*;

public class SortedMapDemo {
    public static void main(String[] args) {

        // vytvoříme mapu
        SortedMap sm = new TreeMap();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Uykoukal");
        sm.put(c1.prijmeni, c1);

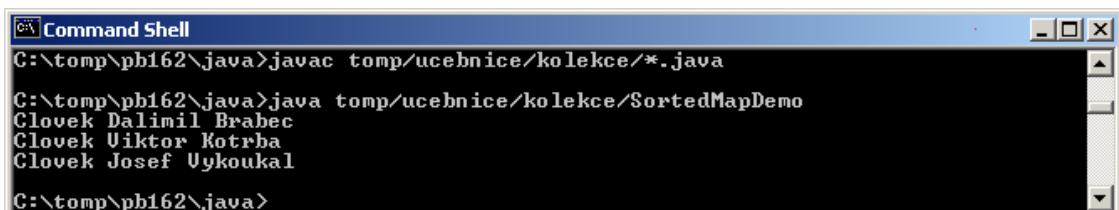
        Clovek c2 = new Clovek("Dalimil", "Brabec");
        sm.put(c2.prijmeni, c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        sm.put(c3.prijmeni, c3);

        // projdi abecedně všechny lidi v mapě
        // proto je třeba získat iterátor nad množinou klíčů
        for(Iterator i = sm.keySet().iterator(); i.hasNext(); ) {
            String prij = (String)i.next();
            Clovek c = (Clovek)sm.get(prij);
            c.vypisInfo();
        }
        static class Clovek {
            String jmeno, prijmeni;
            Clovek (String j, String p) {
                jmeno = j;
                prijmeni = p;
            }
            public void vypisInfo() {
                System.out.println("Clovek "+jmeno+" "+prijmeni);
            }
        }
    }
}

```

Obrázek 8.14. Spuštění funkčního pg. s uspořádanou mapou



```

C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/SortedMapDemo
Clovek Dalimil Brabec
Clovek Viktor Kotrba
Clovek Josef Uykoukal
C:\tomp\pb162\java>

```

## Uspořádaná mapa - příklad s komparátorem

Příklad, kdy jsou klíče uspořádané komparátorem:

### Obrázek 8.15. Vložení účtů do mapy pod uspořádaným klíčem člověka - vlastníka

```
package tomp.ucebnice.kolekce;

import java.util.*;

public class SortedMapComparatorDemo {
    public static void main(String[] args) {

        // vytvoříme mapu
        SortedMap sm = new TreeMap(new ClovekComparator());

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        Ucet u1 = new Ucet(100);
        sm.put(c1, u1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        Ucet u2 = new Ucet(50);
        sm.put(c2, u2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        Ucet u3 = new Ucet(2000);
        sm.put(c3, u3);

        // projdi abecedně všechny vlastníky účtů v mapě
        // proto je třeba získat iterátor nad množinou klíčů
        for(Iterator i = sm.keySet().iterator(); i.hasNext(); ) {
            Clovek majitel = (Clovek)i.next();
            Ucet ucet = (Ucet)sm.get(majitel);
            majitel.vypisInfo();
            System.out.println(" je majitelem uctu se zustatkem "
                               + ucet.zustatek + " Kc");
        }
    }

    static class Ucet {
        double zustatek;
        public Ucet(double z) {
            zustatek = z;
        }
    }
}
```

```

static class Clovek { // nemusí být Comparable
    String jmeno, prijmeni;
    Clovek (String j, String p) {
        jmeno = j;
        prijmeni = p;
    }
    public void vypisInfo() {
        System.out.print("Clovek "+jmeno+" "+prijmeni);
    }
}

static class ClovekComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        // porovnává jen lidi a to podle příjmení
        if (o1 instanceof Clovek && o2 instanceof Clovek) {
            Clovek c1 = (Clovek)o1;
            Clovek c2 = (Clovek)o2;
            return c1.prijmeni.compareTo(c2.prijmeni);
        } else
            throw new IllegalArgumentException(
                "Nelze porovnat objekt typu Clovek s objektem jiného typu");
    }
}
}

```

Obrázek 8.16. Spuštění funkčního pg. s uspořádanou mapou

```

C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/SortedMapComparatorDemo
Clovek Dalimil Brabec je majitelem uctu se zustatkem 50.0 Kc
Clovek Viktor Kotrba je majitelem uctu se zustatkem 2000.0 Kc
Clovek Josef Vykoukal je majitelem uctu se zustatkem 100.0 Kc
C:\tomp\pb162\java>

```

## Starší typy kontejnerů

### Historie

Existují tyto starší typy kontejnerů (-> náhrada):

- *Hashtable* -> *HashMap*, *HashSet* (podle účelu)
- *Vector* -> *List*
- *Stack* -> *List*

Roli iterátoru plnil dříve výčet (*enumeration*) se dvěma metodami:

- *boolean hasMoreElements()*
- *Object nextElement()*

## Srovnání implementací a odkazy

### Srovnání implementací kontejnerů

*Seznamy:*

- na bázi pole (*ArrayList*) - rychlý přímý přístup (přes index)
- na bázi lineárního zřetězeného seznamu (*LinkedList*) - rychlý sekvenční přístup (přes iterátor)

téměř vždy se používá *ArrayList* - stejně rychlý a paměťově efektivnější

*Množiny a mapy:*

- na bázi hašovacích tabulek (*HashMap*, *HashSet*) - rychlejší, ale neuspořádané (lze získat iterátor procházející klíče uspořádaně)
- na bázi vyhledávacích stromů (*TreeMap*, *TreeSet*) - pomalejší, ale uspořádané
- spojení výhod obou - *LinkedHashSet*, *LinkedHashMap* - novinka v Javě 2, v1.4

## Odkazy

Demo      efektivita      práce      kontejnerů      -      Demo      kolekce  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/kolekce/Kolekce.java>]

Velmi podrobné a kvalitní seznámení s kontejnery najdete na Trail: Collections  
[<http://java.sun.com/docs/books/tutorial/collections/index.html>]


---

# Kapitola 9. Výjimky.


## Výjimky

- Výjimky - proč a jak, co to vlastně je výjimka
- Syntaxe bloku s ošetřením (zachycením) výjimky
- „Únik“ výjimky z metody - deklarace metody propouštějící výjimku
- Reakce na výjimku
- Kaskády výjimek
- Kategorizace výjimek (hlídané, běhové, vážné chyby)
- Vlastní typy výjimek, objektová hierarchie výjimek
- Klauzule finally




## Co a k čemu jsou výjimky

- podobně jako v C/C++, Delphi
- výjimky jsou mechanismem, jak psát robustní, spolehlivé programy odolné proti chybám "okolí" - uživatele, systému...
- v Javě jsou výjimky ještě lépe implementovány než v C++ (navíc klauzule finally   
[<http://www.instantweb.com/foldoc/foldoc.cgi?finally>])
- není dobré výjimkami "pokrývat" chyby programu samotného - to je hrubé zneužití

## Výjimky technicky

- *Výjimka* (*Exception*) je objekt třídy `java.lang.Exception`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?java.lang.Exception>]
- Objekty -*výjimky*- jsou vytvářeny (vyvolávány) buďto
  - automaticky běhovým systémem Javy, nastane-li nějaká běhová chyba, např. dělení nulou, nebo
  - jsou vytvořeny samotným programem, zdetekuje-li nějaký chybový stav, na nějž je třeba reagovat - např. do metody je předán špatný argument



- Vzniklý objekt výjimky je předán buďto:
  1. v rámci metody, kde výjimka vznikla - do bloku `catch`   
`[http://www.instantweb.com/foldoc/foldoc.cgi?catch]` -> výjimka je v bloku `catch`   
`[http://www.instantweb.com/foldoc/foldoc.cgi?catch]` tzv. **zachycena**
  2. výjimka "propadne" do nadřazené (volající) metody, kde je buďto v bloku `catch`   
`[http://www.instantweb.com/foldoc/foldoc.cgi?catch]` zachycena nebo opět propadne atd.
- Výjimka tedy "putuje programem" tak dlouho, než je zachycena
- -> pokud není, běh JVM skončí s hlášením o výjimce


## Syntaxe kódu s ošetřením výjimek

Základní syntaxe:

```
try {
    //zde může vzniknout výjimka

} catch (TypVýjimky proměnnáVýjimky) {
    // zde je výjimka ošetřena
    // je možné zde přistupovat k proměnnéVýjimky
}
```


Příklad - Otevření souboru může vyvolat výjimku  
`[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/OtevreniSouboru.java]`

Bloku `try`  `[http://www.instantweb.com/foldoc/foldoc.cgi?try]` se říká *hlídaný blok*, protože výjimky (příslušného hlídaného typu) zde vzniklé jsou zachyceny.

## Syntaxe metody propouštějící výjimku

Pokud výjimka nikde v těle nemůže vzniknout, překladač to zdetekuje a vypíše:

```
... Exception XXX is never thrown in YYYY ...
```



`[http://www.instantweb.com/foldoc/foldoc.cgi?... Exception XXX is never thrown in YYYY ...]`

Příklad s propouštěnou výjimkou -Otevření souboru s propouštěnou výjimkou  
`[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/OtevreniSouboru2.java]`



```
modifikatory návratovýTyp nazevMetody(argumenty) throws TypPropouštěnéVýjimky {
    ... tělo metody, kde může výjimka vzniknout ...
}
```

## Reakce na výjimku

Jak můžeme reagovat?





1. Napravit příčiny vzniku chybového stavu - např. znovu nechat načíst vstup
2. Poskytnout za chybný vstup náhradu - např. implicitní hodnotu
3. Operaci neprovést („vzdát“) a sdělit chybu výše tím, že výjimku „propustíme“ z metody

Výjimková pravidla:

1. Vždy nějak reagujeme! Neignorujeme, nepotlačujeme, tj.
2. blok `catch`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?catch\]](http://www.instantweb.com/foldoc/foldoc.cgi?catch) *nenecháme prázdný*, přinejmenším `e.printStackTrace()`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?e.printStackTrace\(\)\]](http://www.instantweb.com/foldoc/foldoc.cgi?e.printStackTrace())
3. Nelze-li reagovat na místě, propustíme výjimku výše (a popíšeme to v dokumentaci...) - Příklad komplexní reakce na výjimku [\[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyZachytZopakuj.java\]](http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyZachytZopakuj.java)

## Kaskády výjimek

V některých blocích `try`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?try\]](http://www.instantweb.com/foldoc/foldoc.cgi?try) mohou vzniknout výjimky více typů:

- pak můžeme bloky `catch`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?catch\]](http://www.instantweb.com/foldoc/foldoc.cgi?catch) řetězit, viz přechozí příklad: Příklad komplexní reakce na výjimku [\[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyZachytZopakuj.java\]](http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyZachytZopakuj.java)
- Pokud `catch`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?catch\]](http://www.instantweb.com/foldoc/foldoc.cgi?catch) řetězíme, musíme respektovat, že výjimka je zachycena nejbližším příhodným `catch`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?catch\]](http://www.instantweb.com/foldoc/foldoc.cgi?catch)
- Pozor na řetězení `catch`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?catch\]](http://www.instantweb.com/foldoc/foldoc.cgi?catch) s výjimkami typů z jedné hierarchie tříd: pak musí být výjimka z podtřídy (tj. speciálnější) uvedena - zachycována - dříve než výjimka obecnější - Takto ne! [\[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/KaskadaVyjimekSpatne.java\]](http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/KaskadaVyjimekSpatne.java)

## Kategorizace výjimek a dalších chybových objektů

- Všechny objekty výjimek a chybových stavů implementují rozhraní `java.lang.Throwable` - „vyhoditelný“
- Nejčastěji se používají tzv. *hlídané výjimky* (checked exceptions) - to jsou potomci/instance třídy `java.lang.Exception`
- Tzv. *běhové (runtime, nebo též nehlídané, unchecked) výjimky* jsou typu/typu potomka --> `java.lang.RuntimeException` - takové výjimky nemusejí být zachytávány
- *Vážné chyby JVM* (potomci/instance `java.lang.Error`) - obvykle signalizují těžce napravitelné chyby v JVM - např. *Out Of Memory, Stack Overflow...*, ale též např. chybu programátora: `AssertionError`

## Vlastní hierarchie výjimek




- Typy (=třídy) výjimek si můžeme definovat sami, např. viz - Výjimky ve světě chovatelství [<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet.html>]
- bývá zvykem končit názvy tříd - výjimek - na *Exception*

## Klauzule

**finally** 

[<http://www.instantweb.com/foldoc/foldoc.cgi?finally>]

Klauzule (blok) *finally*:

- Může následovat ihned po bloku `try`  [<http://www.instantweb.com/foldoc/foldoc.cgi?try>] nebo až po blocích `catch`  [<http://www.instantweb.com/foldoc/foldoc.cgi?catch>]
- Slouží k "úklidu v každém případě", tj.
  - když je výjimka zachycena blokem `catch`  [<http://www.instantweb.com/foldoc/foldoc.cgi?catch>]
  - i když je výjimka *propuštěna* do volající metody
- Používá se typicky pro uvolnění systémových zdrojů - uzavření souborů, soketů...

Příklad: Úklid se musí provést v každém případě...  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyFinally.java>]

## Odkazy

- Sun Java Tutorial - Lesson: Handling Errors with Exceptions  
[<http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html>]
- demo programy z učebnice - Výjimky  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky>]

---

# Kapitola 10. Vstupy a výstupy v Javě.

## Vstupy a výstupy v Javě




- Koncepce I/O proudů v Javě, skládání (obalování vlastnostmi)
- Práce se soubory a adresáři, třída File
- Binární proudy, třídy InputStream, OutputStream
- Znakové proudy, třídy Reader, Writer
- Serializace objektů

## Koncepce vstupně/výstupních operací v Javě

založeny na v/v proudech

plně **platformově nezávislé**


V/V proudy jsou

- **znakové** (Reader  [http://www.instantweb.com/foldoc/foldoc.cgi?Reader]/Writer   
[http://www.instantweb.com/foldoc/foldoc.cgi?Writer]) nebo
- **binární** (Stream  [http://www.instantweb.com/foldoc/foldoc.cgi?Stream])

koncepovány jako "stavebnice" - lze vkládat do sebe a tím přidávat vlastnosti, např.

```
is = new InputStream(...);  
bis = new BufferedInputStream(is);
```


Téměř vše ze vstupních/výstupních tříd a rozhraní je v balíku java.io.

počínaje J2SDK1.4 se rozvíjí alternativní balík - java.nio   
[http://www.instantweb.com/foldoc/foldoc.cgi? java.nio ] (*New I/O*)

Bližší viz dokumentace API balíků java.io  
[http://java.sun.com/j2se/1.4.2/docs/api/java/io/package-summary.html],  
[http://java.sun.com/j2se/1.4.2/docs/api/java/nio/package-summary.html].  
java.io  
java.nio

## Práce se soubory

 [http://www.instantweb.com/foldoc/foldoc.cgi?java.io]

základem je třída `java.io.File`  [http://www.instantweb.com/foldoc/foldoc.cgi?java.io.File] - nositel jména souboru, jakási "brána" k fyzickým souborům na disku.

používá se jak pro soubory, tak adresáře, linky i soubory identifikované UNC jmény (\\počítač\adresář...)

opět plně platformově nezávislé

na odstínění odlišností jednotlivých systémů souborů lze použít vlastností (uvádíme jejich hodnoty pro JVM pod systémem MS Windows):

- `File.separatorChar`   [http://www.instantweb.com/foldoc/foldoc.cgi?File.separatorChar] - jako `char`  [http://www.instantweb.com/foldoc/foldoc.cgi?char]
- `File.separator`  [http://www.instantweb.com/foldoc/foldoc.cgi?File.separator] - jako `String`  [http://www.instantweb.com/foldoc/foldoc.cgi?String]
- `File.pathSeparatorChar`   [http://www.instantweb.com/foldoc/foldoc.cgi?File.pathSeparatorChar] - jako `char`  [http://www.instantweb.com/foldoc/foldoc.cgi?char]
- `File.pathSeparator`  [http://www.instantweb.com/foldoc/foldoc.cgi?File.pathSeparator] - jako `String`  [http://www.instantweb.com/foldoc/foldoc.cgi?String]
- `System.getProperty("user.dir")`  [http://www.instantweb.com/foldoc/foldoc.cgi?System.getProperty("user.dir")] - adresář uživatele, pod jehož UID je proces JVM spuštěn

## Třída

**File** 

[http://www.instantweb.com/foldoc/foldoc.cgi?File]

Vytvoření konstruktorem - máme několik možností:

|                                                            |                                                                                 |
|------------------------------------------------------------|---------------------------------------------------------------------------------|
| <code>new File(String filename)</code>                     | vytvoří v aktuálním adresáři soubor s názvem <i>filename</i>                    |
| <code>new File(File baseDir, String filename)</code>       | vytvoří v adresáři <i>baseDir</i> soubor s názvem <i>filename</i>               |
| <code>new File(String baseDirName, String filename)</code> | vytvoří v adresáři <i>se jménem baseDirName</i> soubor s názvem <i>filename</i> |
| <code>new File(URL url)</code>                             | vytvoří soubor se (souborovým - file:) URL <i>url</i>                           |

Testy existence a povahy souboru:

boolean      test na existenci souboru (nebo adresáře)

**exists()**

boolean **is-**      test, zda jde o soubor (tj. ne adresář)

**File()**

() test, zda jde o adresář

Test práv ke čtení/zápisu:

boolean **canRead()** test, zda lze soubor číst  
boolean **canWrite()** test, zda lze do souboru zapisovat

## Třída

**File**

[<http://www.instantweb.com/foldoc/foldoc.cgi?File>] (2)

Vytvoření souboru nebo adresáře:

boolean **createNewFile()** (pro soubor) vrací `true` [http://www.instantweb.com/foldoc/foldoc.cgi?true], když se podaří *soubor* vytvořit

boolean **mkdir()** (pro adresář) vrací `true` [http://www.instantweb.com/foldoc/foldoc.cgi?true], když se podaří adresář vytvořit

boolean **mkdirs()** navíc si dotvoří i příp. neexistující adresáře na cestě

Vytvoření dočasného (temporary) souboru:

static File **createTempFile(String prefix, String suffix)** vytvoří dočasný soubor ve standardním pro to určeném adresáři (např. `c:\temp` [http://www.instantweb.com/foldoc/foldoc.cgi?c:\temp]) s uvedeným prefixem a sufixem názvu

static File **createTempFile(String prefix, String suffix, File directory)** dtto, ale vytvoří dočasný soubor v adr. `directory` [http://www.instantweb.com/foldoc/foldoc.cgi?directory]

boolean **delete()** zrušení souboru nebo adresáře

Přejmenování (ne přesun mezi adresáři!):

boolean **renameTo(File dest)** přejmenuje soubor nebo adresář

## Třída

**File**



## [<http://www.instantweb.com/foldoc/foldoc.cgi?File>] (3)

Další vlastnosti:

long **len-**      délka (velikost) souboru v bajtech  
**gth()**

() čas poslední modifikace v ms od začátku éry - podobně jako systémový čas vrácený `System.currentTimeMillis()`.

String jen jméno souboru (tj. poslední část cesty)

(  
String **getName()** celá cesta k souboru i se jménem


**getPath()**  
String **getAbsolutePath()** absolutní cesta k souboru i se jménem

**getAbsolutePath()**


() adresář, v němž je soubor nebo adresář obsažen

Bliže viz dokumentace API třídy File [<http://java.sun.com/j2se/1.4.2/docs/api/java/io/File.html>].

## Práce s adresáři

Klíčem je opět třída File  [<http://www.instantweb.com/foldoc/foldoc.cgi?File>] - použitelná i pro adresáře


Jak např. získat (filtrovaný) seznam souborů v adresáři?


pomocí metody `File[] listFiles(FileFilter ff)`  [<http://www.instantweb.com/foldoc/foldoc.cgi?File>] `listFiles(FileFilter ff)` nebo podobné

`File[] listFiles(FilenameFilter fnf)`:

`FileFilter` je rozhraní s jedinou metodou `boolean accept(File pathname)`, obdobně `FilenameFilter`, viz Popis API `java.io.FilenameFilter` [<http://java.sun.com/j2se/1.4/docs/api/java/io/FilenameFilter.html>]

## Práce s binárními proudy

Vstupní jsou odvozeny od abstraktní třídy `InputStream`  [<http://www.instantweb.com/foldoc/foldoc.cgi?InputStream>]

Výstupní jsou odvozeny od abstraktní třídy `OutputStream`  [<http://www.instantweb.com/foldoc/foldoc.cgi?OutputStream>]

## Vstupní binární proudy


Uvedené metody, kromě `abstract byte read()`, nemusejí být nutně v neabstraktní podtřídě překryty.


|                                                   |                                                                                         |
|---------------------------------------------------|-----------------------------------------------------------------------------------------|
| <code>void close()</code>                         | uzavře proud a uvolní příslušné zdroje (systémové "file handles" apod.)                 |
| <code>void mark(int readlimit)</code>             | poznačí si aktuální pozici (později se lze vrátit zpět pomocí <code>reset()</code> )... |
| <code>boolean markSupported()</code>              | ...ale jen když platí tohle                                                             |
| <code>abstract int read()</code>                  | přečte bajt (0-255 pokud OK; jinak -1, když už není možné přečíst)                      |
| <code>int read(byte[] b)</code>                   | přečte pole bajtů                                                                       |
| <code>int read(byte[] b, int off, int len)</code> | přečte pole bajtů se specifikací délky a pozice plnění pole b                           |
| <code>void reset()</code>                         | vrátí se ke značce nastavené metodou <code>mark(int)</code>                             |
| <code>long skip(long n)</code>                    | přeskočí zadaný počte bajtů                                                             |

## Důležité neabstraktní třídy odvozené od `InputStream`

[<http://www.instantweb.com/foldoc/foldoc.cgi?InputStream>]

`java.io.FilterInputStream` - je базová třída k odvozování všech vstupních proudů přidávajících vlastnost/schopnost filtrovat poskytnutý vstupní proud.

Příklady filtrů (ne všechny jsou v `java.io`  [<http://www.instantweb.com/foldoc/foldoc.cgi?java.io>!]):

|                                                |                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>BufferedInputStream</code>               | proud s vyrovnávací pamětí (je možno specifikovat její optimální velikost)                                                                                                                                                                                                                                      |
| <code>java.util.zip.CheckedInputStream</code>  | proud s kontrolním součtem (např. CRC32)                                                                                                                                                                                                                                                                        |
| <code>javax.crypto.CipherInputStream</code>    | proud dešifrující data ze vstupu                                                                                                                                                                                                                                                                                |
| <code>DataInputStream</code>                   | má metody pro čtení hodnot primitivních typů, např. <code>readFloat()</code> <br>[ <a href="http://www.instantweb.com/foldoc/foldoc.cgi?float readFloat()">http://www.instantweb.com/foldoc/foldoc.cgi?float readFloat()</a> ] |
| <code>java.security.DigestInputStream</code>   | počítá současně i haš (digest) čtených dat, použitý algoritmus lze nastavit                                                                                                                                                                                                                                     |
| <code>java.util.zip.InflaterInputStream</code> | dekomprimuje (např. GZIPem) zabalený vstupní proud (má ještě specializované podtřídy)                                                                                                                                                                                                                           |
| <code>LineNumberInputStream</code>             | doplňuje informaci o tom, ze kterého řádku vstupu čteme (zavrhovaná - <i>deprecated</i> - třída)                                                                                                                                                                                                                |
| <code>ProgressMonitorInputStream</code>        | přidává schopnost informovat o průběhu čtení z proudu                                                                                                                                                                                                                                                           |
| <code>PushbackInputStream</code>               | do proudu lze data vracet zpět                                                                                                                                                                                                                                                                                  |


## Další vstupní proudy

Příklad rekonstrukce objektů ze souborů

```
FileInputStream istream = new FileInputStream("t.tmp");
ObjectInputStream p = new ObjectInputStream(istream);
int i = p.readInt();
String today = (String)p.readObject();
Date date = (Date)p.readObject();
istream.close();
```

|                                                 |                                                                       |
|-------------------------------------------------|-----------------------------------------------------------------------|
| <code>vax.sound.sampled.AudioInputStream</code> | vstupní proud zvukových dat                                           |
| <code>ByteArrayInputStream</code>               | proud dat čtených z pole bajtů                                        |
| <code>PipedInputStream</code>                   | roura napojená na "protilehlý" <code>PipedOutputStream</code>         |
| <code>SequenceInputStream</code>                | proud vzniklý spojením více podřízených proudů do jednoho virtuálního |
| <code>ObjectInputStream</code>                  | proud na čtení serializovaných objektů                                |

## Práce se znakovými proudy





základem je abstraktní třída `Reader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Reader>], konkrétními implementacemi jsou:

- `BufferedReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?BufferedReader>], `CharArrayReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?CharArrayReader>], `InputStreamReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?InputStreamReader>], `PipedReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?PipedReader>], `StringReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?StringReader>]
- `LineNumberReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?LineNumberReader>], `FileReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?FileReader>], `PushbackReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?PushbackReader>]

## Výstupní proudy



nebudeme důkladně probírat všechny typy

principy:

- jedná se o protějšky k vstupním proudům, názvy jsou konstruovány analogicky (např. `FileReader`  [<http://www.instantweb.com/foldoc/foldoc.cgi?FileReader>] -> `FileWriter`  [<http://www.instantweb.com/foldoc/foldoc.cgi?FileWriter>])
- místo generických metod `read`  [<http://www.instantweb.com/foldoc/foldoc.cgi?read>] mají `write(...)`  [[http://www.instantweb.com/foldoc/foldoc.cgi?write\(...\)](http://www.instantweb.com/foldoc/foldoc.cgi?write(...))]



Příklady:

`PrintStream` poskytuje metody pro pohodlný zápis hodnot primitivních typů a řetězců - příkladem

jsou `System.out`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?System.out\]](http://www.instantweb.com/foldoc/foldoc.cgi?System.out) a  
`System.err`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?System.err\]](http://www.instantweb.com/foldoc/foldoc.cgi?System.err)


`PrintWriter` poskytuje metody pro pohodlný zápis hodnot primitivních typů a řetězců

## Konverze: znakové <-> binární proudy

Ze vstupního binárního proudu `InputStream`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?InputStream\]](http://www.instantweb.com/foldoc/foldoc.cgi?InputStream) (čili každého) je možné vytvořit znakový  
`Reader`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?Reader\]](http://www.instantweb.com/foldoc/foldoc.cgi?Reader) pomocí

```
// nejprve binární vstupní proud - toho kódování znaků nezajímá
InputStream is = ...
```

```
// znakový proud isr
// použije pro dekódování standardní znakovou sadu
Reader isr = new InputStreamReader(is);
```



```
// sady jsou definovány v balíku java.nio  \[http://www.instantweb.com/foldoc/foldoc.cgi?java.nio\]
Charset chrs = java.nio.Charset.forName("ISO-8859-2");
```




```
// znakový proud isr2
// použije pro dekódování jinou znakovou sadu
Reader isr2 = new InputStreamReader(is, chrs);
```

Podporované názvy znakových sad naleznete na webu IANA Charsets  
[\[http://www.iana.org/assignments/character-sets\]](http://www.iana.org/assignments/character-sets).

Obdobně pro výstupní proudy - lze vytvořit `Writer` z `OutputStream`.

## Serializace objektů

- nebudeme podrobně studovat, zatím stačí vědět, že:
  - **serializace objektů** je postup, jak z objektu vytvořit sekvenci bajtů persistentně uložitelnou na paměťové médium (disk) a později restaurovatelnou do podoby výchozího javového objektu.
  - **deserializace** je právě zpětná rekonstrukce objektu
- aby objekt bylo možno serializovat, musí implementovat (prázdné) rozhraní `java.io.Serializable`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?java.io.Serializable\]](http://www.instantweb.com/foldoc/foldoc.cgi?java.io.Serializable)
- proměnné objektu, které nemají být serializovány, musí být označeny modifikátorem - klíčovým slovem - `transient`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?transient\]](http://www.instantweb.com/foldoc/foldoc.cgi?transient)

- pokud požaduje "speciální chování" při de/serializaci, musí objekt definovat metody
  - `private void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?private> void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException]
  - `private void writeObject(java.io.ObjectOutputStream stream) throws IOException`  [<http://www.instantweb.com/foldoc/foldoc.cgi?private> void writeObject(java.io.ObjectOutputStream stream) throws IOException]
- metody:
  - `DataOutputStream.writeObject(Object o)`   
[[http://www.instantweb.com/foldoc/foldoc.cgi?DataOutputStream.writeObject\(Object o\)](http://www.instantweb.com/foldoc/foldoc.cgi?DataOutputStream.writeObject(Object o))]

## Odkazy

Tutoriály k Java I/O: kapitola z Sun Java Tutorial [<http://java.sun.com/docs/books/tutorial/essential/io/>]

Demo programy na serializaci (z učebnice): Serializace objektů  
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/serializace/>]

---

# Kapitola 11. Pokročilejší témata. Použití javadoc. Distribuce aplikací - jar. Generické datové typy. Novinky v Javě 5.

## Dokumentace aplikací

- Dokumentace javových programů, dokumentace API
- Typy komentářů - dokumentační komentáře
- Generování dokumentace
- Značky javadoc

## Dokumentace javových programů

Základním a standardním prostředkem je tzv. *dokumentace API*

- Dokumentace je *naprosto nezbytnou součástí* javových programů.
- Rozlišujeme dokumentaci např. instalační, systémovou, uživatelskou, programátorskou...

Zde se budeme věnovat především dokumentaci programátorské, určené těm, kdo budou náš kód využívat ve svých programech, rozšiřovat jej, udržovat jej. Programátorské dokumentaci se říká *dokumentace API* (apidoc, apidocs).


Při jejím psaní dodržujeme tato pravidla:

- Dokumentujeme především *veřejné* (public) a *chráněné* (protected) prvky (metody, proměnné). Ostatní dle potřeby.
- Dokumentaci píšeme *přímo do zdrojového kódu* programu ve *speciálních dokumentačních komentářích* vpisovaných *před příslušné prvky* (metody, proměnné).
- Dovnitř metod píšeme jen *pomocné komentáře* pro programátory (nebo pro nás samotné).

## Typy komentářů




Podobně jako např. v C/C++:

|                                                                                   |                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| řádkové                                                                           | od značky // do konce řádku, nepromítnou se do dokumentace API                                                                                                                                           |
| blokové                                                                           | začínají /*  [http://www.instantweb.com/foldoc/foldoc.cgi?/*] pak je text komentáře, končí */ na libovolném počtu řádků |
| dokumentační                                                                      | od značky /** po značku */ může být opět na libovolném počtu řádků                                                                                                                                       |
| Každý další řádek může začínat mezerami či *, hvězdička se v komentáři neprojeví. |                                                                                                                                                                                                          |


## Kde uvádíme dokumentační komentáře

Dokumentační komentáře uvádíme:

- Před *hlavičkou třídy* - pak komentuje třídu jako celek.
- Před *hlavičkou metody nebo proměnné* - pak komentuje příslušnou metodu nebo proměnnou.
- Celý balík (package) je možné komentovat *speciálním samostatným HTML souborem* package-summary.html [http://www.instantweb.com/foldoc/foldoc.cgi?package-summary.html] uloženým v adresáři balíku.

## Generování dokumentace


Dokumentace má standardně podobu HTML stránek (s rámy i bez)


Dokumentace je generována nástrojem javadoc  
[http://www.instantweb.com/foldoc/foldoc.cgi?javadoc] z

1. dokumentačních komentářů
2. i ze samotného zdrojového textu

Lze tedy (základním způsobem) dokumentovat i program bez vložených komentářů!

Chování javadoc [http://www.instantweb.com/foldoc/foldoc.cgi?javadoc] můžeme změnit


1. volbami (options) při spuštění,
2. použitím jiného tzv. doclet [http://www.instantweb.com/foldoc/foldoc.cgi?doclet]u, což je třída implementující potřebné metody pro generování komentářů.

Princip generování ze zdrojových textů pomocí speciálních doclet  
[<http://www.instantweb.com/foldoc/foldoc.cgi?doclet>]ů se dnes používá i po jiné než dokumentační účely - např. pro generátory zdrojových kódu aplikací EJB apod.

## Značky

javadoc

[<http://www.instantweb.com/foldoc/foldoc.cgi?javadoc>]

javadoc [<http://www.instantweb.com/foldoc/foldoc.cgi?javadoc>] můžeme podrobněji instruovat pomocí značek vkládaných do dokumentačních komentářů, např.:

|             |                                                                     |
|-------------|---------------------------------------------------------------------|
| @author     | specifikuje autora API/programu                                     |
| @version    | označuje verzi API, např. "1.0"                                     |
| @deprecated | informuje, že prvek je zavrhováný                                   |
| @exception  | popisuje informace o výjimce, kterou metoda propouští ("vyhazuje")  |
| @param      | popisuje jeden parametr metody                                      |
| @since      | uvedeme, od kdy (od které verze pg.) je věc podporována/přítomna    |
| @see        | uvedeme odkaz, kam je také doporučeno nahlédnout (související věci) |

## Příklad zdrojového textu se značkami javadoc

Zdrojový text třídy *Window*:

```
/**
 * Klasse, die ein Fenster auf dem Bildschirm repräsentiert
 * Konstruktor zum Beispiel:
 * <pre>
 * Window win = new Window(parent);
 * win.show();
 * </pre>
 *
 * @see awt.BaseWindow
 * @see awt.Button
 * @version 1.2 31 Jan 1995
 * @author Bozo the Clown
 */
class Window extends BaseWindow {
    ...
}
```

Příklad dokumentačního komentáře k proměnné:


```
/**
 * enthält die aktuelle Anzahl der Elemente.
 * muss positiv und kleiner oder gleich der Kapazität sein
 **/
protected int count;
```

Tyto a další příklady a odkazy lze vidět v původním materiálu JavaStyleGuide des IGE [http://www.iam.unibe.ch/~scg/Resources/PSE/2001/WWW/projektHandbuch/codeInspections/JavaStyleGuide.html], odkud byly ukázky převzaty.

## Spouštění

javadoc 


[http://www.instantweb.com/foldoc/foldoc.cgi?javadoc]

- javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]  [http://www.instantweb.com/foldoc/foldoc.cgi?javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]]
- možné volby:
  - -help  [http://www.instantweb.com/foldoc/foldoc.cgi?-help], -verbose   
[http://www.instantweb.com/foldoc/foldoc.cgi?-verbose]
  - -public  [http://www.instantweb.com/foldoc/foldoc.cgi?-public], -protected   
[http://www.instantweb.com/foldoc/foldoc.cgi?-protected], -package   
[http://www.instantweb.com/foldoc/foldoc.cgi?-package], -private   
[http://www.instantweb.com/foldoc/foldoc.cgi?-private] - specifikuje, které prvky mají být v dokumentaci zahrnuty (implicitně: -protected   
[http://www.instantweb.com/foldoc/foldoc.cgi?-protected])
  - -d destinationdirectory  [http://www.instantweb.com/foldoc/foldoc.cgi?-d destinationdirectory ] - kam se má dok. uložit
  - -doctitle title  [http://www.instantweb.com/foldoc/foldoc.cgi?-doctitle title ] - titul celé dokumentace



## Příklady

Zdroják s dokumentačními komentáři - Komentáře  
[http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet/Clovek.java]

Ukázkové spuštění

javadoc  [http://www.instantweb.com/foldoc/foldoc.cgi?javadoc]



```
javadoc -classpath . -d apidocs svet
```





vytvoří dokumentaci tříd z balíku svet  [\[http://www.instantweb.com/foldoc/foldoc.cgi?svet\]](http://www.instantweb.com/foldoc/foldoc.cgi?svet) do adresáře apidocs  [\[http://www.instantweb.com/foldoc/foldoc.cgi?apidocs\]](http://www.instantweb.com/foldoc/foldoc.cgi?apidocs)

## Distribuce aplikací

- Distribuční archívy .jar
- Vytvoření archívu, metainformace
- Spustitelné archívy


## Distribuce aplikací

Distribucí nemyslíme použití nástroje typu "InstallShield"..., ale spíše něčeho podobného tar   
[\[http://www.instantweb.com/foldoc/foldoc.cgi?tar\]](http://www.instantweb.com/foldoc/foldoc.cgi?tar) ZIP   
[\[http://www.instantweb.com/foldoc/foldoc.cgi?ZIP\]](http://www.instantweb.com/foldoc/foldoc.cgi?ZIP)u

- Java na sbalení množiny souborů zdrojových i přeložených (.class   
[\[http://www.instantweb.com/foldoc/foldoc.cgi?.class\]](http://www.instantweb.com/foldoc/foldoc.cgi?.class)) nabízí nástroj jar   
[\[http://www.instantweb.com/foldoc/foldoc.cgi?jar\]](http://www.instantweb.com/foldoc/foldoc.cgi?jar).
- Sbalením vznikne soubor (archív) .jar  [\[http://www.instantweb.com/foldoc/foldoc.cgi?.jar\]](http://www.instantweb.com/foldoc/foldoc.cgi?.jar) formátově podobný ZIP  [\[http://www.instantweb.com/foldoc/foldoc.cgi?ZIP\]](http://www.instantweb.com/foldoc/foldoc.cgi?ZIP)u (obvykle *je* to ZIP formát), ale nemusí být komprimován.
- Kromě souborů obsahuje i metainformace (tzv. *MANIFEST*)
- Součástí archívu nejsou jen .class soubory, ale i další zdroje, např. *obrázky, soubory s národními variantami řetězců, zdrojové texty programu, dokumentace...*

## Spuštění

**jar**   
[\[http://www.instantweb.com/foldoc/foldoc.cgi?jar\]](http://www.instantweb.com/foldoc/foldoc.cgi?jar)

- `jar {ctxu} [vfm0M] [jar-file] [manifest-file] [-C dir] files`   
[\[http://www.instantweb.com/foldoc/foldoc.cgi? jar {ctxu} \[vfm0M\] \[jar-file\] \[manifest-file\] \[-C dir\] files\]](http://www.instantweb.com/foldoc/foldoc.cgi?jar%20%7Bctxu%7D%20%5Bvfm0M%5D%20%5Bjar-file%5D%20%5Bmanifest-file%5D%20%5B-C%20dir%5D%20files)
- .

-  [http://www.instantweb.com/foldoc/foldoc.cgi?c] - vytvoří archiv
- t  [http://www.instantweb.com/foldoc/foldoc.cgi?t] - vypíše obsah archívu
- x  [http://www.instantweb.com/foldoc/foldoc.cgi?x] - extrahuje archiv
- u  [http://www.instantweb.com/foldoc/foldoc.cgi?u] - aktualizuje obsah archívu
- volby:
  - v  [http://www.instantweb.com/foldoc/foldoc.cgi?v] - verbose
  - 0  [http://www.instantweb.com/foldoc/foldoc.cgi?0] - soubory nekomprimuje
  - f  [http://www.instantweb.com/foldoc/foldoc.cgi?f] - pracuje se se souborem, ne se "stdio"
  - m  [http://www.instantweb.com/foldoc/foldoc.cgi?m] - přibálí metainformace z manifest-file
  - le  [http://www.instantweb.com/foldoc/foldoc.cgi?manifest-file]
- parametr files  [http://www.instantweb.com/foldoc/foldoc.cgi?files] uvádí, které soubory se sbalí - i nejavové (např. typicky dokumentace API - HTML, datové soubory)

## Volby

**[http://www.instantweb.com/foldoc/foldoc.cgi?jar]**

jar 

Volby JAR lze vypsát i spuštěním jar bez parametrů:

**Obrázek 11.1. Volby nástroje JAR**

```

C:\tomp\pb162\java>jar
Usage: jar <ctxu>[vfm0Ml] [jar-file] [manifest-file] [-C dir] files ...
Options:
  -c    create new archive
  -t    list table of contents for archive
  -x    extract named (or all) files from archive
  -u    update existing archive
  -v    generate verbose output on standard output
  -f    specify archive file name
  -m    include manifest information from specified manifest file
  -0    store only; use no ZIP compression
  -M    do not create a manifest file for the entries
  -i    generate index information for the specified jar files
  -C    change to the specified directory and include the following file
If any file is a directory then it is processed recursively.
The manifest file name and the archive file name needs to be specified
in the same order the 'm' and 'f' flags are specified.

Example 1: to archive two class files into an archive called classes.jar:
  jar cvf classes.jar Foo.class Bar.class
Example 2: use an existing manifest file 'mymanifest' and archive all the
  files in the foo/ directory into 'classes.jar':
  jar cvfm classes.jar mymanifest -C foo/ .

C:\tomp\pb162\java>

```

## jar [<http://www.instantweb.com/foldoc/foldoc.cgi?jar>] - příklad

Vezměme následující zdrojový text třídy *JarDemo* v balíku *tomp.ucebnice.jar*, tj. v adresáři *c:\tomp\pb162\java\tomp\ucebnice\jar* 

[<http://www.instantweb.com/foldoc/foldoc.cgi?c:\tomp\pb162\java\tomp\ucebnice\jar>]:


Obrázek 11.2. Třída *JarDemo*

```

package tomp.ucebnice.jar;

public class JarDemo {
    public static void main(String[] args) {
        System.out.println("Spustena trida JarDemo!");
    }
}


```

Vytvoříme archiv se všemi soubory z podadresáře *tomp/ucebnice/jar*   
[<http://www.instantweb.com/foldoc/foldoc.cgi?tomp/ucebnice/jar>] (s volbou *c* - create, *v* - verbose, *f* - do souboru):

Obrázek 11.3. Vytvoření archívu se všemi soubory z podadresáře


## tomp/ucebnice/jar

```
Command Shell
C:\tomp\pb162\java>jar cfv jardemo.jar tomp/ucebnice/jar/*. *
added manifest
adding: tomp/ucebnice/jar/JarDemo.class(in = 449) (out= 306)(deflated 31%)
adding: tomp/ucebnice/jar/JarDemo.java(in = 164) (out= 130)(deflated 20%)
adding: tomp/ucebnice/jar/JarDemo.java.bak(in = 0) (out= 0)(stored 0%)
C:\tomp\pb162\java>_
```

Vzniklý `.jar`  [<http://www.instantweb.com/foldoc/foldoc.cgi?.jar>] soubor lze prohlédnout/rozbalit také běžným nástrojem typu unzip, gunzip, WinZip, PowerArchiver nebo souborovým managerem typu Servant Salamander...

### Obrázek 11.4. .jar archiv v okně PowerArchiveru



Tento archiv rozbalíme v adresáři `/temp`  [<http://www.instantweb.com/foldoc/foldoc.cgi?/temp>] následujícím způsobem:


### Obrázek 11.5. Vybalení všech souborů z archívu

```
Command Shell
C:\temp>jar xfv /tomp/pb162/java/jardemo.jar *. *
C:\temp>_
```

## Rozšíření

`.jar` 

## [<http://www.instantweb.com/foldoc/foldoc.cgi?jar>] archívů

Formáty vycházející z JAR  [<http://www.instantweb.com/foldoc/foldoc.cgi?JAR>]:

- pro webové aplikace - .war  [<http://www.instantweb.com/foldoc/foldoc.cgi?.war>]
- pro enterprise (EJB) aplikace - .ear  [<http://www.instantweb.com/foldoc/foldoc.cgi?.ear>]

liši se podrobnějším předepsáním adresářové struktury a dalšími povinnými metainformacemi

## Tvorba spustitelných archívů

Vytvoříme jar  [<http://www.instantweb.com/foldoc/foldoc.cgi?jar>] s manifestem obsahujícím tento řádek:

Main-Class: *NázevSpouštěnéTřídy*  [[http://www.instantweb.com/foldoc/foldoc.cgi? Main-Class: \*NázevSpouštěnéTřídy\* \]](http://www.instantweb.com/foldoc/foldoc.cgi?Main-Class:NázevSpouštěnéTřídy)

poté zadáme:

```
java -jar NázevBalíku.jar
```

a spustí se metoda `main`  [<http://www.instantweb.com/foldoc/foldoc.cgi?main>] třídy *NázevSpouštěnéTřídy*.

## Vytvoření spustitelného archívu - příklad

Nejprve vytvoříme soubor manifestu. Příklad jeho obsahu:

**Obrázek 11.6. Soubor manifestu**



Následně zabalíme archív s manifestem:

**Obrázek 11.7. Zabalení archívu s manifestem**



```
Command Shell
C:\tomp\pb162\java>jar cfvm jardemo.jar jardemo-manifest.txt tomp/ucebnice/jar/*
.*
added manifest
adding: tomp/ucebnice/jar/JarDemo.class(in = 449) (out= 306)(deflated 31%)
adding: tomp/ucebnice/jar/JarDemo.java(in = 164) (out= 130)(deflated 20%)
adding: tomp/ucebnice/jar/JarDemo.java.bak(in = 0) (out= 0)(stored 0%)
C:\tomp\pb162\java>
```

## Spuštění archívu - příklad

Spuštění aplikace zabalené ve spustitelném archívu je snadné:



```
java -jar jardemo.jar
```

a spustí se metoda *main* třídy *tomp.ucebnice.jar.JarDemo*:

Obrázek 11.8. Spuštění aplikace z archívu

```
Command Shell
added manifest
adding: tomp/ucebnice/jar/JarDemo.class(in = 449) (out= 306)(deflated 31%)
adding: tomp/ucebnice/jar/JarDemo.java(in = 164) (out= 130)(deflated 20%)
adding: tomp/ucebnice/jar/JarDemo.java.bak(in = 0) (out= 0)(stored 0%)
C:\tomp\pb162\java>java -jar jardemo.jar
Spustena trida JarDemo!
C:\tomp\pb162\java>
```


Další příklad spuštění jar   
[<http://www.instantweb.com/foldoc/foldoc.cgi?jar>]


- `jar tfv svet.jar | more`  [<http://www.instantweb.com/foldoc/foldoc.cgi?jar> tfv svet.jar | more]
- vypíše po obrazovkách obsah (listing) archívu svet.jar   
[<http://www.instantweb.com/foldoc/foldoc.cgi?svet.jar>]


## Generické typy


### Generické datové typy (generics)

Pokud si vezmeme anglicko-český slovník, zjistíme, že v překladu to znamená něco obecně použitelného, tedy mohli bychom použít termínu *zobecnění*. A přesně tím generics jsou -- zobecněním.

Jak již víme, struktura tříd v Javě má společného předka, třídu `Object` 

[<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. Tato skutečnost zjednodušuje implementaci ne-jednoho programu -- potřebujeme-li pracovat s nějakými objekty, o kterých tak úplně nevíme, co jsou zač, můžeme využít společného předka a pracovat s ním. Každý objekt v programu je totiž i instancí třídy `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. To v praxi umožňuje například snadnou implementaci spojových seznamů, hashovacích tabulek, ale například i využití reflexe.


Jakkoliv je společný předek jistě výhodou, přináší i některé obtíže. Uvažujme opět (spojový) seznam. Pracujeme-li v programu s nějakým, víme jenom, že je to seznam objektů a nic více. Samozřejmě si můžeme bokem pamatovat, že tam ukládáme jenom řetězce, ale to nic nemění na tom, že runtime systému je srdečně jedno, jaké objekty do seznamu vkládáme. Stejně tak, chceme-li z tohoto seznamu číst, získáváme zase jenom obecné objekty, které musíme explicitně přetypovat na řetězec (tj. třídu `String` 

[<http://www.instantweb.com/foldoc/foldoc.cgi?String>]). A opět -- ačkoliv jako programátoři víme, co v seznamu *má být*, obecně si nemůžeme být jisti, zda to tam skutečně *je*. Což vede buď k tomu, že se budeme ptát, zda získávaný objekt je skutečně řetězcem a až poté jej přetypujeme nebo budeme „riskovat“ runtime výjimku `ClassCastException` 


[<http://www.instantweb.com/foldoc/foldoc.cgi?ClassCastException>]. Jak vidíme, nevýhody jsou a to docela významné.

Řešením v této situaci je příchod Javy verze 1.5, která mimo jiné přináší i zmiňovaná *generics*. Opět využijeme našeho spojového seznamu. Vraťme se zpět k jeho definici. Co od něj vlastně požadujeme?


- Aby byl seznamem čehokoliv a tak byl universální (což je stav popsán výše)?
- Nebo aby to byl seznam nějakého obecného, předem nedefinovaného typu a umožnil tento typ při vytvoření instance určit?

Jelikož předchozí otázky jsou řečnické, odpověď následuje ihned. Samozřejmě, že druhá uvedená možnost je lepší (už proto, že má veškerou sílu první možnosti, stačí nadefinovat, aby tím obecným typem byl `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]). Ve zbytku textu si tedy ukážeme, jak toho *generics* docílují a jak je používat.

## Základní syntaxe

V úvodu jsme se lehce zmínili o spojovém seznamu. Nyní si budeme ukazovat příklady na obecném seznamu (rozhraní `java.util.List` 

[<http://www.instantweb.com/foldoc/foldoc.cgi?java.util.List>]).

Takhle vypadá deklarace `List`  [<http://www.instantweb.com/foldoc/foldoc.cgi?List>] bez použití *generics*:

```
public interface List {
    ...
}
```


A takhle s nimi:

```
public interface List <E> {  
    ...  
}
```

Jak vidíme, úvod je velmi jednoduchý. Do špičatých závorek pouze umístíme symbol, kterým říkáme, že seznam bude obsahovat prvky *E* (předem neznámého) typu.

Zde uděláme malou odbočku -- je doporučováno používat velké, jednopísmenné deklarace generics. Toto písmeno by zároveň mělo vystihovat použití resp. význam takového zobecnění. Tedy *T* je typ, *E* je prvek (element) a tak podobně

## Jednoduché využití v metodách

Pouhá deklarace u jména třídy resp. rozhraní samozřejmě nemůže stačit. Zjednodušeně řečeno, zdrojový kód využívající generics musí typ *E* použít všude tam, kde by dříve použil obecný `Object`   
[<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. To jest například místo

```
Object get(int index);
```


se použije

```
E get(int index);
```

Co jsme nyní udělali? Touto definicí jsme řekli, že metoda `get` vrací pouze objekty, které jsou typu *E* na místo libovolného objektu, což je přesně to, co od generics vyžadujeme. Všimněte si, že nyní už s *E* pracujeme jako s jakoukoliv jinou třídou nebo rozhraním.

Totožně postupujeme i u metod, které do seznamu prvky typu *E* přidávají. Viz

```
boolean add(E o);
```

Dovolím si další malou poznámku na okraj -- výše zmíněné metody by samozřejmě mohly pracovat s typem `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. Překladač by proti tomu nic nenamítal, nicméně očekávaná funkcionalita by byla pryč.

## První příklad použití

Nyní tedy máme seznam, který při použití bude obsahovat nějaké prvky typu *E*. Nyní chceme takový seznam použít někde v našem kódu a užívat si výhod generics. Vytvoříme jej následovně:




```
List<String> = new ArrayList<String>();
```

Použití je opět jednoduché a velmi intuitivní. Nyní následují dva příklady demonstrující výhody generics (první je napsán „postaru“).

```
Object number = new Integer(2);
List numbers = new ArrayList();
numbers.add(new Integer(1));
numbers.add(number);
Number n = (Number) numbers.get(0);
Number o = (Number) numbers.get(1);
```


```
Object number = 2;
List<Number> numbers = new ArrayList<Number>();
numbers.add(1);
numbers.add((Number) number);
Number n = numbers.get(0);
Number o = numbers.get(1);
```

Jak vidíme v horním příkladu, do seznamu lze vložit libovolný objekt (byť zde jsme měli „štěstí“ a bylo to číslo) a při získávání objektů se spoléháme na to, že se jedná o číslo. Níže naopak nelze obecný objekt vložit, je nutné jej explicitně přetypovat na číslo, teprve poté překladač kód zkompileje. Podotkněme, že


 [http://www.instantweb.com/foldoc/foldoc.cgi?Number] přetypovali například `String`  [http://www.instantweb.com/foldoc/foldoc.cgi?String], program se také přeloží, ale v okamžiku zavolání takového příkazu se logicky vyvolá výjimka `ClassCastException`  [http://www.instantweb.com/foldoc/foldoc.cgi?ClassCastException]. Získání čísel ze seznamu je ovšem přímočaré -- stačí pouze zavolat metodu `get`, která má správný návratový typ.

Povšimněte si rovněž použití další vlastnosti nové Javy, tzv. *autoboxingu*, kdy primitivní typ je automaticky převeden na odpovídající objekt (a vice versa).

## Cyklus foreach


Přestože konstrukce cyklu `for`  [http://www.instantweb.com/foldoc/foldoc.cgi?for] patří svou povahou jinam, zmiňujeme se o nich zde, u dynamických struktur - kontejnerů, neboť se převážně používá k iterování (procházení) prvků seznamů, množin a dalších struktur. Obecný tvar cyklu "foreach" je syntaktickou variantou běžného "for":

```
for (TypRidiciPromenne ridici_promenna : dyn_struktura) {  
    // co se dela pro kazdou hodnotu ze struktury...  
}
```

V následujícím příkladu jsou v jednotlivých průchodech cyklem `for`  [http://www.instantweb.com/foldoc/foldoc.cgi?for] postupně ze seznamu vybírány a do řídicí proměnné `e` přiřazovány všechny jeho prvky (objekty).


```
for (Object e : seznam) {  
    System.out.println(e);  
}
```

## Žolíci (wildcards)

V předchozích částech jsme se seznámili s hlavní myšlenkou generics a její realizací, a sice nahrazení konkrétního nadtypu (většinou `Object`  [http://www.instantweb.com/foldoc/foldoc.cgi?Object]) typem obecným. Nicméně tohle samo o sobě je velmi omezující a nedostačující. Nyní se tedy ponoříme hlouběji do tajů generics.

Představme si následující situaci. V programu chceme mít seznam, kde budou jako prvky různé jiné seznamy. První nápad, jak jej nadeklarovat může být třeba tento:

```
List<List<Object>> seznamSeznamu;
```

Na první pohled se to zdá být bez chyby. Máme seznam, kam budeme vkládat jiné seznamy a jelikož každý seznam musí obsahovat instance třídy `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>], můžeme tam vložit libovolný seznam, tedy třeba i náš `List<Number>`. Nicméně tato úvaha je chybná. Uvažujme následující kód:


```
List<Number> cisla = new ArrayList<Number>();
List<Object> obecny = cisla;
obecny.add("Ja nejsem cislo");
```


Jak vidíme, „něco je špatně.“ To, že se pokoušíme přiřadit do seznamu objektů `obecny` řetězec `"Ja nejsem cislo"` je přece naprosto v pořádku, do seznamu objektů můžeme skutečně vložit cokoliv. V tom případě ale musí být špatně přiřazení na druhém řádku. To znamená, že seznam čísel *není* seznamem objektů! Zde je vidět rozdíl oproti „klasickému“ uvažování v mezích dědičnosti. Přečtěte si pozorně následující větu a pokuste se pochopit její význam.

*Do seznamu, který obsahuje nejvýše čísla lze vkládat pouze objekty, které jsou alespoň čísla.*

Z toho vyplývá, že je nelegální přiřazovat objekt „seznam čísel“ do objektu „seznam objektů.“ Tedy, vrátíme-li se k našemu příkladu se seznamem seznamů, vidíme, proč byla naše úvaha chybná. Do námi definovaného seznamu totiž lze ukládat pouze seznamy objektů a ne libovolné seznamy. Jak tedy docílíme kýženého jevu? K tomuto účelu nám generics poskytují nástroj zvaný *žolík*, anglicky *wildcard*, který se zapisuje jako `?`. Vraťme se nyní k předchozímu příkladu:

```
List<Number> cisla = new ArrayList<Number>();
List<?> obecny = cisla;           // tohle je OK
obecny.add("Ja nejsem cislo");    // tohle nelze prelozit
```


Jak je již v komentáři kódu naznačeno, poslední řádek neprojde překladačem. Proč? Protože pomocí `List<?>`  [<http://www.instantweb.com/foldoc/foldoc.cgi?List<?>>] říkáme, že `obecny` je seznamem *neznámých* prvků. A jelikož nevíme, jaké prvky v seznamu jsou, nemůžeme do něj ani *žádné* prvky přidávat. Jedinou výjimkou je „žádný“ prvek, totiž `null`, který lze přidat kamkoliv. Mírně filosoficky řečeno, *null není ničím a tak je zároveň vším*.

Naopak, ze seznamu neznámých objektů můžeme samozřejmě prvky číst, neboť každý prvek je určitě alespoň instancí třídy `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. Ukážeme si praktické použití žolíku.

```
public static void tiskniSeznam(List<?> seznam) {
    for (Object e : seznam) {
        System.out.println(e);
    }
}
```

Nyní si představme, že chceme metodu, která udělá z nějakého seznamu čísel jeho sumu. Uvažujme tedy následující (a pomeňme možné přetečení nebo podtečení rozsahu `double`):

```
public static double suma(List<Number> cisla) {
    double result = 0;
    for (Number e : cisla) {
        result += e.doubleValue();
    }
    return result;
}
```


Opět, metoda se jeví jako bezproblémová. Nic ale není tak jednoduché, jak by se mohlo zdát. Nyní zkusíme uvažovat bez příkladu. Představme si, že máme seznam celých čísel, u kterého chceme provést sumu. Jistě není sporu o tom, že celá čísla jsou zároveň obecná čísla a přesto seznam `List<Integer>`  [<http://www.instantweb.com/foldoc/foldoc.cgi?List<Integer>>] nelze použít jako parametr výše deklarované metody z naprosto stejného důvodu, kvůli kterému nešlo říci, že seznam objektů je seznam čísel.





Samozřejmě je tu opět řešení. Zkusme nejdříve uvažovat selským rozumem. Výše jsme říkali, že místo *seznamu objektů* chceme *seznam neznámých prvků*. Nyní jsme v podobné situaci, pouze se nacházíme na jiném místě v hierarchii tříd. Zkusme tedy obdobnou úvahu použít i zde. Nechceme *seznam čísel* nýbrž *seznam neznámých prvků, které jsou nejvýše čísla*. Nyní je již pouze třeba ozřejmit syntaxi takové „úvahy“.

```
public static double suma(List<? extends Number> cisla) {
    ...
}
```

Toto použití žolíku má uplatnění i v samotném rozhraní `List<E>` a sice v metodě „přidej vše“. Zamyslete se nad tím, proč tomu tak je.

```
boolean addAll(Collection<? extends E> c);
```

Uvědomte si prosím následující -- prostý žolík je vlastně „zkratka“ pro „neznámý prvek rozšiřující `Object`“  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>].

Ač by se tak mohlo zdát, možnosti *wildcards* jsme ještě nevyčerpali. Představme si situaci, kdy potřebujeme, aby možnou hodnotou byla instance třídy, která je v hierarchii mezi třídou specifikovanou naším obecným prvkem *E* a třídou `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. Pokud přemýšlíte, k čemu je něco takového dobré, představte si, že máte množinu celých čísel, které chcete seřadit. Jak lze taková čísla třídit? Například obecně podle hodnoty metody `hashCode()`, tedy na úrovni třídy `Object`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Object>]. Nebo jako obecné číslo, tj. na úrovni třídy `Number`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Number>]. A konečně i jako celé číslo na úrovni třídy `Integer`  [<http://www.instantweb.com/foldoc/foldoc.cgi?Integer>]. Skutečně, níže již jít nemůžeme, protože libovolné zjemnění této třídy například na celá kladná čísla by nemohlo třídit obecná celá čísla.

Následující příklad demonstruje syntaxi a použití popsané konstrukce

```
public TreeMap(Comparator<? super K> c);
```

Jedná se o konstruktor stromové mapy, tj. mapy klíč/hodnota, která je navíc seříděna podle klíče. Nyní opět trochu odbočíme a podíváme se, jak vypadá deklarace obecného rozhraní seříděné mapy.

```
public interface SortedMap<K,V> extends Map<K,V> {...
```

Máme zde nový prvek -- je-li třeba použít více nezávislých obecných typů, zapíšeme je opět do „zobáčků“ jako seznam hodnot oddělených čárkou. Povšimněte si opět mnemotechniky -- *K* je *key* (klíč),



*V* je *value* (hodnota). Je-li to třeba, je možné použít i žolíků. Viz následující příklad konstruktorů naší staré známé stromové mapy.





```
public TreeMap(Map<? extends K, ? extends V> m);  
public TreeMap(SortedMap<K, ? extends V> m);
```

## Generické metody

Tato část bude relativně krátká a stručná, poněvadž pro používání *generics* a žolíků platí stále stejná pravidla. Generickou metodou rozumíme takovou, která je parametrizována alespoň jedním obecným typem, který nějakým způsobem „váže“ typy proměnných a/nebo návratové hodnoty metody.

Představme si například, že chceme statickou metodu, která přenesení prvky z pole nějakého typu přidá hodnoty do seznamu s prvky téhož typu.

```
static <T> void arrayToList(T[] array, List<T> list) {  
    for (T o : array) {  
        list.add(o);  
    }  
}
```

Zde narážíme na malou záludnost. Ve skutečnosti nemusí být seznam *list* téhož typu, stačí, aby jeho typ byl nadtřídou typu pole *array*. To se může jevit jako velmi matoucí, ovšem pouze do té chvíle, dokud si neuvědomíme, že pokud máme např. pole celých čísel, tj. `Integer`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?Integer\]](http://www.instantweb.com/foldoc/foldoc.cgi?Integer) a seznam obecných čísel `Number`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?Number\]](http://www.instantweb.com/foldoc/foldoc.cgi?Number), pak platí, že pole prvků typu `Integer`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?Integer\]](http://www.instantweb.com/foldoc/foldoc.cgi?Integer) JE polem prvků typu `Number`  [\[http://www.instantweb.com/foldoc/foldoc.cgi?Number\]](http://www.instantweb.com/foldoc/foldoc.cgi?Number)! Skutečně, zde se dostáváme zpět ke klasické dědičnosti a nesmí nás mást pravidla, která platí pro obecné typy ve třídách.

## Generics metody vs. wildcards

Jak již bylo zmíněno, je žádoucí, aby typ použitý u generické metody spojoval alespoň dva parametry nebo parametr a návratovou hodnotu. Následující příklad demonstruje nesprávné použití generické metody.

```
public static <T, S extends T> void copy(List<T> destination, List<S> source
```

Příklad je syntakticky bezproblémový, dokonce jej lze i přeložit a bude fungovat dle očekávání. Nicméně správný zápis by měl být následující.

```
public static <T> void copy(List<T> destination, List<? extends T> source);
```

Zde je již vidět požadovaná vlastnost --  $T$  spojuje dva parametry metody a přebytečné  $S$  je nahrazené žolíkem. V prvním příkladu si všimněte zápisu  $S \text{ extends } T$ . Ukazuje další možnou deklaraci generics.


## Pole

Při deklaraci pole nelze použít parametrizovanou třídu, pouze třídu s žolíkem, který není vázaný (nebo bez použití žolíku). Tj. jediná správná deklarace je následující:

```
List<?>[] pole = new List<?>[10];
```

Parametrizovanou třídu v seznamu nelze použít z toho důvodu, že při vkládání prvků do nich runtime systém kontroluje pouze *typ* vkládaného prvku, nikoliv už to, zda využívá generics a zda tento odpovídá deklarovanému typu. To znamená, že měli bychom například pole seznamů, které obsahují pouze řetězce, mohli bychom do něj bez problémů vložit pole čísel. To by samo o sobě nic nezpůsobilo, ovšem mohlo by dojít k „přeměně“ typu generics, čímž by se seznam čísel „proměnil“ na seznam řetězců, což by bylo špatně.


## Vícenásobná vazba generics

Uvažujme následující metodu (bez použití generics), která vyhledává maximální prvek nějaké kolekce. Navíc platí, že prvky kolekce musí implementovat rozhraní `Comparable`, což, jak lze snadno nahlédnout, není syntaxí vůbec podchyceno a tudíž zavolání této metody může vyvolat výjimku `ClassCastException`  [<http://www.instantweb.com/foldoc/foldoc.cgi?ClassCastException>].

```
public static Object max(Collection c);
```

Nyní se pokusíme vymyslet, jak zapsat tuto metodu za použití generics. Chceme, aby prvky kolekce implementovali rozhraní Comparable. Podíváme-li se na toto rozhraní, zjistíme, že je též parametrizované generics. Potřebujeme tedy takovou instanci, která je schopná porovnat libovolné třídy v hierarchii nad třídou, která bude prvkem vstupní kolekce. První pokus, jak zapsat požadované.

```
public static <T extends Comparable<? super T>> T max(Collection<T> c);
```

Tento zápis je relativně OK. Metoda správně vrátí proměnnou stejného typu, jaký je prvkem v kolekci, dokonce i použití Comparable  [http://www.instantweb.com/foldoc/foldoc.cgi?Comparable ] je správné. Nicméně, pokud bychom se zajímali o signaturu metody po „výmazu“ generics, dostaneme následující.

```
public static Comparable max(Collection c);
```

To neodpovídá signatuře metody výše. Využijeme tedy vícenásobné vazby.

1

```
public static <T extends Object & Comparable<? super T>> T max (Collection<T>
```

Nyní, po „výmazu“ má již metoda správnou signaturu, protože v úvahu se bere první zmíněná třída. Obecně lze použít více vazeb pro generics, například chceme-li, aby obecný prvek byl implementací více rozhraní.

## Závěr

V článku jsme se seznámili se základními i některými pokročilými technikami použití *generics*. Tato technologie má i další využití, například u reflexe. Tohle však již překračuje rámec začátečnického seznámení s Javou.

Celý článek vychází z materiálů, které jsou volně k dispozici na oficiálních stránkách Javy firmy Sun

---

<sup>1</sup>V materiálu, ze kterého čerpám, je navíc `Collection<? extends T>` Domnívám se ovšem, že metoda zmíněná v tomto článku má stejnou funkcionalitu. Pokud se někomu podaří nalézt protipříklad, budu rád.

[<http://java.sun.com/>], zejména pak z *Generics in the Java Programming Language* od Gilada Brachy. Některé příklady v této stati jsou převzaty ze zmíněného článku.

## Další změny v Javě 5



### Poznámka

Psáno se svolením autora podle článku P. Adámka "Co se děje v Javě" (publikováno ve sborníku SLT2004)

- zrychlení startu aplikací
- výčtové typy
- ostatní vylepšení

## Zrychlení startu aplikací

- Mezi pravidelné změny v každé nové verzi jazyka Java patří zvýšení rychlosti; v tomto směru Java 1.5 nikterak nevybočuje. Kromě několika změn v garbage collectoru a virtuálním stroji je asi nejvýznamnějším prvkem vlastnost nazvaná *class data sharing*. Tento mechanismus urychluje start virtuálního stroje a šetří paměť v případě několika souběžně spuštěných virtuálních strojů. Princip spočívá v tom, že se při instalaci JRE (nebo Java SDK) spustí virtuální stroj a pak je paměťová oblast vyhrazená pro read-only data uložena do souboru na disk. Při každém dalším spuštění JVM se tato oblast namapuje do paměti, což start výrazně urychlí. Pokud běží více virtuálních strojů naráz, tato oblast je sdílená.
- Mechanismus funguje pouze pro klientskou verzi JVM; serverová verze podporována není, což je ale pochopitelné. Serverové aplikace jsou většinou vytvářeny jako vícevláknové v rámci jednoho virtuálního stroje a běží nepřetržitě.

## Výčtový typ

Další věc, která doposud v Javě chybí, je typ `enum`. To se doposud řešilo několika způsoby, od definování konstant typu `int` (což postrádá jakoukoliv typovou kontrolu) až po užívání typově bezpečného řešení založeného na definici třídy pro každý výčtový typ, navrženého v publikaci J. Blocha. Toto řešení ve své nejjednodušší variantě vypadá takto:

```
public class Color {
    private Color() {}

    public static final Color red    = new Color();
    public static final Color green  = new Color();
    public static final Color blue   = new Color();
}
```

```
public static final Color yellow = new Color();  
}
```

Právě na tomto principu je založená implementace výčtového typu, která je od verze 5 přímou součástí jazyka Java. Výčtový typ se zde ale definuje mnohem snadněji:

```
public enum Color { red, green, blue, yellow };
```

Pro úplnost je nutné poznamenat, že takto definovaný výčtový typ disponuje mnohem širšími možnostmi než předchozí varianta. Dále viz dokumentace API k Java 5 [<http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Enum.html>].

## Ostatní vylepšení Javy 5

|                              |                                                                                                                                                                                                                                                |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Metadata                     | Tento nový prvek Jazyka Java umožňuje zdrojový kód rozšířit o doplňkové informace deklarativního charakteru; uplatnění nalezne zejména v J2EE, kde eliminuje nutnost vytváření spousty zbytečného kódu a psaní <i>deployment deskriptorů</i> . |
| Formátování vstupu a výstupu | Nyní bude v Javě konečně možné používat mocné formátování výstupu funkcí <code>fprint</code> , známou z jazyků C/C++. Kromě toho jsou rozšířeny možnosti zpracování vstupu.                                                                    |
| Proměnný počet parametrů     | Toto je další prvek známý z jazyků C/C++; osobně jej považuji za nepříliš důležitý a někteří lidé tvrdí, že byl zaveden jenom kvůli metodě <i>printf</i> (viz předchozí bod).                                                                  |
| Nový <i>Look and Feel</i>    | Ve Windows přibyl nový vzhled pro knihovnu Swing, který vypadá jako Windows XP, v Linuxu pak vzhled, který vypadá jako GTK.                                                                                                                    |
| Rozšířená podpora pro XML    | Za tímto téměř marketingovým pojmem se skrývá mnoho drobných ale velice užitečných vylepšení. Patří mezi ně plná podpora XML 1.1 a jmenných prostorů, XML Schémat, SAX 2.0.1, XSLT a XSLTC, DOM level 3, XPath, a další.                       |
| Monitorování a správa        | Umožní monitorovat a ovládat virtuální stroj pomocí rozhraní JMX.                                                                                                                                                                              |

---

# Kapitola 12. Informace k testům.

## Písemky

Obecné komentáře ke všem písemkám

- písemky se píší přímo do počítače
- výsledkem je přeložitelný program; nepřeložitelný program může být hodnocen 0 body
- odevzdává se zdrojový text
- podkladem pro písemku může být výsledek předchozí(ch) úloh(y)

## První písemka

- **12 bodů**
- píše se během cvičení, čas tedy max 1 hod (spíše 50 min).
- odevzdá se dle pokynů cvičícího
- předmětem bude využít předem známé API (existující třídy)
- v případě omluvené nepřítomnosti je možné psát v náhradním termínu, který je následující cvičení (neurčí-li cvičící jinak)
- písemku zadává a hodnotí cvičící

## Druhá písemka

- **18 bodů**
- píše se během cvičení, čas tedy max 1 hod (spíše 50 min).
- odevzdá se dle pokynů cvičícího
- v případě omluvené nepřítomnosti je možné psát v náhradním termínu, který je následující cvičení (neurčí-li cvičící jinak)
- písemku zadává a hodnotí cvičící

## Třetí (zkoušková) písemka

- **30 bodů**
- píše se ve zkouškovém období, čas i s kontrolou max 1:45 hod.
- přihlašuje se na ni jako na **zkouškový termín**
- odevzdá se dle pokynů zadávajícího
- písemku zadává a hodnotí vyučující předmětu nebo osoba pověřená - ne nutně váš cvičící

---

# Kapitola 13. Dodatky - Co je nového v Javě 1.5.

## Další změny v Javě 5



### Poznámka

Psáno se svolením autora podle článku P. Adámka "Co se děje v Javě" (publikováno ve sborníku SLT2004)

- zrychlení startu aplikací
- výčtové typy
- ostatní vylepšení

## Zrychlení startu aplikací

- Mezi pravidelné změny v každé nové verzi jazyka Java patří zvýšení rychlosti; v tomto směru Java 1.5 nikterak nevybočuje. Kromě několika změn v garbage collectoru a virtuálním stroji je asi nejvýznamnějším prvkem vlastnost nazvaná *class data sharing*. Tento mechanismus urychluje start virtuálního stroje a šetří paměť v případě několika souběžně spuštěných virtuálních strojů. Princip spočívá v tom, že se při instalaci JRE (nebo Java SDK) spustí virtuální stroj a pak je paměťová oblast vyhrazená pro read-only data uložena do souboru na disk. Při každém dalším spuštění JVM se tato oblast namapuje do paměti, což start výrazně urychlí. Pokud běží více virtuálních strojů naráz, tato oblast je sdílená.
- Mechanismus funguje pouze pro klientskou verzi JVM; serverová verze podporována není, což je ale pochopitelné. Serverové aplikace jsou většinou vytvářeny jako vícevláknové v rámci jednoho virtuálního stroje a běží nepřetržitě.

## Výčtový typ

Další věc, která doposud v Javě chybí, je typ `enum`. To se doposud řešilo několika způsoby, od definování konstant typu `int` (což postrádá jakoukoliv typovou kontrolu) až po užívání typově bezpečného řešení založeného na definici třídy pro každý výčtový typ, navrženého v publikaci J. Blocha. Toto řešení ve své nejjednodušší variantě vypadá takto:

```
public class Color {  
    private Color() {};
```



```
public static final Color red    = new Color();
public static final Color green  = new Color();
public static final Color blue   = new Color();
public static final Color yellow = new Color();
}
```

Právě na tomto principu je založená implementace výčtového typu, která je od verze 5 přímou součástí jazyka Java. Výčtový typ se zde ale definuje mnohem snadněji:

```
public enum Color { red, green, blue, yellow };
```

Pro úplnost je nutné poznamenat, že takto definovaný výčtový typ disponuje mnohem širšími možnostmi než předchozí varianta. Dále viz dokumentace API k Java 5 [<http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Enum.html>].

## Ostatní vylepšení Javy 5

|                              |                                                                                                                                                                                                                                                |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Metadata                     | Tento nový prvek Jazyka Java umožňuje zdrojový kód rozšířit o doplňkové informace deklarativního charakteru; uplatnění nalezne zejména v J2EE, kde eliminuje nutnost vytváření spousty zbytečného kódu a psaní <i>deployment deskriptorů</i> . |
| Formátování vstupu a výstupu | Nyní bude v Javě konečně možné používat mocné formátování výstupu funkcí <code>fprint</code> , známou z jazyků C/C++. Kromě toho jsou rozšířeny možnosti zpracování vstupu.                                                                    |
| Proměnný počet parametrů     | Toto je další prvek známý z jazyků C/C++; osobně jej považuji za nepříliš důležitý a někteří lidé tvrdí, že byl zaveden jenom kvůli metodě <i>printf</i> (viz předchozí bod).                                                                  |
| Nový <i>Look and Feel</i>    | Ve Windows přibyl nový vzhled pro knihovnu Swing, který vypadá jako Windows XP, v Linuxu pak vzhled, který vypadá jako GTK.                                                                                                                    |
| Rozšířená podpora pro XML    | Za tímto téměř marketingovým pojmem se skrývá mnoho drobných ale velice užitečných vylepšení. Patří mezi ně plná podpora XML 1.1 a jmenných prostorů, XML Schémat, SAX 2.0.1, XSLT a XSLTC, DOM level 3, XPath, a další.                       |
| Monitorování a správa        | Umožní monitorovat a ovládat virtuální stroj pomocí rozhraní JMX.                                                                                                                                                                              |