PB162 - Programování v jazyku Java

PB162 - Programování v jazyku Java

Table of Contents

Chapter 1. Programování v jazyce Java

Tomáš Pitner

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 ???.
- **18 bodů - druhá písemka** řešení praktické úlohy přímo u počítače, v době cvičení, na konci výuky v semestru (prosinec). Bližší info viz ???.
- **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 ???.
- **Celkem 100 bodů**

Note

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

Note

docházka do cvičení je povinná, ale netvoří součást bodového hodnocení

Kritéria hodnocení předmětu

K ukončení zkouškou 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 potřebujete:

- 60 bodů

Obsah (sylabus) předmětu

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

Cvičení

Cvičení se konají 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 jen *hodinová*, účast na přednáškách je proto žádoucí
- předpokládá se i jistý netriviální podíl *práce mimo cvičení*
- odměnou vám, kromě získaných znalostí a dovedností, bude i *jednokreditové navýšení na celkem 4 kr.+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]
- www TP: http://www.fi.muni.cz/~tomp

  materiály PB162-Java (IS): https://is.muni.cz/auth/el/1433/podzim2004/PB162/ [https://is.muni.cz/auth/el/1433/podzim2004/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/podzim2004/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
  - 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 [???], 2000) - pro pokročilé
  - a další, viz např.http://www.vltava.cz
  - Joshua Bloch: **Java efektivně 57 zásad softwarového experta**, Grada Publishing [???]
  - 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]

Chapter 2. Úvod do jazyka Java

Tomáš Pitner

Úvod do jazyka Java

- Úvod, srovnání s jinými, oblasti použití Javy
- Distribuce, instalace, použití Java SDK
- Životní cyklus programu v Javě
- Základní praktické úkony - *velmi podrobně!!!*

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 úrovně přístupu k hostitelskému systému pomocí tzv. Security Manageru [http://www.securingjava.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.intellij.com/]

Java pro programátora (2)

Konkrétní možnosti:

- v Javě se dobře píší *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/])
- *Výukové účely* (nahrazuje Pascal jako referenční jazyk)
- Další přenositelné aplikace - např. klientské/desktopové

Hlavní domény Javy (2)

- webové aplikace (servlety, JSP - konkurence proprietárním ASP, SSI, ... a pomalým CGI)
- zpracování semistrukturovaných dat (XML)
- přenositelné aplikace s GUI
- aplikace distribuované po síti (applety nebo Java Web Start)

Javová platforma

Javovou platformu tvoří:

- **Java Virtual Machine**
- **překladač** (přístupný např. příkazem **javac**) a další vývojové nástroje
- **Java Core API** (základní knihovna tříd)

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 přidáván nové klíčové slovo "assert", v1.5 přidá enum a další.

Specifikace a implementace Javy

- **Specifikace** Javy (tzv. "Editions") - např.: *Java 2 Standard Edition, v1.4*
- **Implementace** Javy ("Development Kits" nebo "Runtime Environments") - např.: *Java 2 Software Development Kit, v1.4.2* - obsahuje vývojové nástroje
- *Java 2 Runtime Enviroment, v1.4* - obsahuje jen **běhové prostředí** pro spouštění hotových přeložených pg.

Verze Javy

- hrubé členění - na verze "*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 dojde až teď se změnou pojmenovávacího schématu

Verze Javy - nově

- hrubé členění - na verze *Java 1.x.y* (před Java 2 v1.5) a *Java 5.z*

Aktuální verze

Stav k září 2004:

- Java 2 Standard Edition v1.4.2 pro všechny platformy je stabilní verzí
- Java 2 Standard Edition v1.5 neboli *Java 5* je k dispozici jako RC (Release Candidate), tj. je tzv. feature-complete (=nebude se nic přidávat, jen se vychytají chyby).
- Ostrá verze Java 5 bude dostupná v Q4 2004.

- aktuálně vždy na webu java.sun.com [http://java.sun.com]

Získání distribuce Javy

- používání Javy pro běžný vývoj (i komerční) je zdarma
- redistribuce javového **vývojového prostředí** je dovolena pouze s **licencí** od Sunu
- redistribuce javového **běhového 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] (pro Windows, Solaris, Linux)
- 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 :-))

Obsah vývojové distribuce Javy

- **Vývojové nástroje (Development Tools)** v `bin` -- určené k vývoji, spouštění, ladění a dokumentování programů v Javě.
- **Běhové prostředí Javy (Java Runtime Environment)** se nalézá v `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` jsou další knihovny nutné pro běh vývojových nástrojů.
- **Ukázkové applety a aplikace (Demo Applets and Applications)** v `demo`. Příklady zahrnují i zdrojový kód.

Obsah vývojové distribuce Javy (2)

- **Hlavičkové soubory pro C (C header Files)** - v `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.jar`.

- **Dokumentace (Documentation)** - v podadresáři `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` soubory umístěné v podadresáři `bin`

- **`java`** - spouštěč (přeloženého bajtkódu)
- **`javac`** - překladač (`.java` -> `.class`)
- **`javadoc`** - generátor dokumentace API
- **`jar`** - správce archivů JAR (sbalení, rozbalení, výpis)
- **`jdb`** - debugger
- **`appletviewer`** - referenční prostředí pro spouštění appletů

Pomocné nástroje ve vývojové distribuci

- **`javah`** - generátor hlavičkových souborů pro C
- **`javap`** - disassembler bajtkódu (např. pro ruční optimalizace, hledání chyb)

Základní životní cyklus javového programu

- Program sestává z jedné (ale obvykle více) **tříd** (class)
- Zdrojový kód každé veřejně přístupné třídy je umístěn v jednom souboru (`NazevTridy.java`)
- Postup:
  - vytvoření zdrojového textu (libovolným editorem čistého textu) -> **`Pokus.java`**
  - překlad (nástrojem `javac`) **`Pokus.java`** -> **`Pokus.class`**
  - spuštění, např. **`java Pokus`**
- překládá se `javac název souboru se třídou` (včetně přípony `.java`!!!)
- spouští se vždy udáním `java` a názvu třídy (bez přípony `.class`!!!)

Struktura javového programu

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

Demo "Ahoj!"

Zdrojový kód v souboru `tomp\ucebnice\Pozdrav.java`

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

Třída `Pozdrav` je umístěna do balíku `tomp.ucebnice` ->

její zdrojový soubor musí být uložen v podadresáři `tomp\ucebnice`.

Překlad a spuštění "Ahoj!"

Překlad

1. Máme nainstalován J2SDK 1.4.2
2. Jsme v adresáři `c:\devel\pb162`, v něm je podadresář `tomp\ucebnice`, v něm je soubor `Pozdrav.java`
3. Spustíme *překlad* `javac tomp\ucebnice\Pozdrav.java`
4. Je-li program správně napsán, přeloží se "mlčky"
5. (výsledný `.class` soubor bude v témže adresáři jako zdroj)

Spuštění

1. Poté spustíme *program* Pozdrav: `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`) se budou hledat v odpovídajících podadresářích aktuálního adresáře (adresáře `.`)
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 4.2.2 [http://pspad.zde.cz] (dostupný zdarma, instalovaný na všech Win strojích v učebnách na FI)

Překlad "Ahoj!" ("for dummies")

Překlad překladačem **javac** (úspěšný, bez hlášení překladače)

Spuštění "Ahoj!" ("for dummies")

Spuštění voláním **java**

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` (řetězec)
- předávají se při spuštění z příkazového řádku do pole `String[] args`

Metoda `main` nevrací žádnou hodnotu - návratový typ je vždy(!) `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)` musejí obsahovat i adresář `JAVA_HOME\bin`

Praktické informace (aneb co je vhodné udělat)

Systémové proměnné by měly obsahovat:

- `JAVA_HOME=kořenový adresář instalace` Javy, např. `JAVA_HOME=c:\j2sdk1.4.2`
- `CLASSPATH=cesty ke třídám` (podobně jako v `PATH` jsou cesty ke spustitelným souborům), např. `CLASSPATH=c:\devel\pb162`

Úloha do cvičení 1.

...Zde bude zadání úlohy č. 1...

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

Chapter 3. Úvod do objektového programování

Tomáš Pitner

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 - jakk 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)

Co je třída a objekt?

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

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

- např. všechny objekty třídy `Clovek` mají vlastnost `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` má vlastnost `jmeno`
- Objekt `panProfesor` typu `Clovek` má vlastnost `jmeno` s hodnotou `"Václav Klaus"`.

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

Příklad - deklarace třídy Clovek

- deklarujme 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. vytvořme instanci - objekt - typu `Clovek`
  2. vypišme informace o něm

Příklad - použití třídy Clovek (1)

Mějme deklarovánu třídu `Clovek`

Metoda `main` v následujícím programu:

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

```
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: pouze deklarace (tj. určení typu) proměnné `ales` - bude typu `Clovek`.

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.

Proměnné - deklarace

Položky `jmeno` a `rokNarozeni` v předchozím příkladu jsou **proměnné** objektu `Clovek`.

Jsou deklarovány v těle deklarace třídy `Clovek`.

Deklarace proměnné objektu má tvar:

```
modifikátory Typ jméno;
```

např.:

```
protected int rokNarozeni;
```

Proměnné - datový typ

Výše uvedená proměnná `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. Podrobněji viz ???

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;
```

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

```
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 písemkách.

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 prvniho objektu
        Clovek petr = new Clovek("Petr Svoboda", 1968); // vytvoření druheho objekt
        System.out.println(ales.jmeno); // přístup k (čtení) proměnné - prvnímu obj
        System.out.println(petr.jmeno); // přístup k (čtení) proměnné - druhému obj
    }
}
```

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

Vytváření objektů

Ve výše uvedených příkladech jsme objekty vytvářeli voláními **new Clovek(...)** bezděčně jsme tak použili

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

Metody

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

- podporgram (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 pdobně 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: typParametru nazevFormalnihoParametru, ...`

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

Rozšiřme 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`), jakmile

- dokončí poslední příkaz v těle metody nebo
- dospěje k příkazu `return`

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

- Vrácenou hodnotu musíme uvést za příkazem **return**. V tomto případě tedy nesmí return chybět!
- Typ vrácené hodnoty musíme *v hlavičce metody deklarovat*.
- Nevrací-li metoda nic, pak musíme namísto typu vracené hodnoty psát `void`.

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

Konstruktory

Co a k čemu jsou konstruktory?

- Konstruktury 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` 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` - *to už vůbec ne*!!!)
- mohou mít parametry
- mohou volat konstruktor rodičovské třídy - ale jen jako svůj první příkaz

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 `Ucet` přetížíme metodu `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).

Shrnutí

Objekty:

- jsou instance "své" třídy

- vytváříme je operátorem **new** - voláním konstruktoru
- vytvořené objekty ukládáme do proměnné stejného typu (nebo typu předka či implementovaného rozhraní - o tom až později)

Odkazy na objekty

Deklarace proměnné objektového typu ještě žádný objekt nevytváří.

To se děje až příkazem - operátorem - **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;**) 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);**

```
...
  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;
```

Příklad - upravený `Ucet` s metodou `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
   }
```

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. Ttakové metody a proměnné nazýváme **statické** a označujeme v deklaraci modifikátorem `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 int kolikMamLidi() {
      return pocetLidi;
   }
   ...
}
```

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