Seminář Java

II

Rekapitulace

- Java je *case sensitive*
- Zdrojový kód (soubor .java) obsahuje jednu veřejnou třídu
- Třídy jsou organizovány do balíků
- Hierarchie balíků odpovídá hierarchii adresářů
- `http://java.sun.com` – distribuce, dokumentace, API, ...
- Příklady ze seminářů jsou na webových stránkách
- Diskuzní fórum `vutbr.fit.courses.ija`

Co je třída a objekt?

Třída

- 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

- Objekt je jeden konkrétní jedinec (reprezentant, entita) příslušné třídy
- pro konkrétní objekt nabývají vlastnosti deklarované třídou konkrétních hodnot

Vlastnosti objektu

Vlastnosti objektů (proměnné i metody) je třeba deklarovat.

- proměnné
  - jsou nositeli "pasivních" vlastností; jakýchsi atributů, charakteristik objektů
  - de facto jde o datové hodnoty svázané (zapouzdřené) v objektu
- metody
  - jsou nositeli "výkonných" vlastností; "dovedností" objektů
  - de facto jde o funkce (procedury) pracující (převážně) nad proměnnými objektu

Deklarace třídy

```
modifikátory class názevTřídy [extends, implements]
{
      tělo třídy
      // deklarace proměnných objektu
      // deklarace metod
}
```

Např.:

```
public class Ucet
{
}
```

Modifikátory

- `public`
- `private`
- `protected`
- *žádný*

Deklarace proměnné objektu

Deklarace proměnné objektu má tvar:

```
modifikátory Typ jméno;
```

např.:

```
protected double castka;
```

Jmenné konvence

- 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

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

Deklarace metody

```
modifikátory typVrácenéHodnoty
                          nazevMetody ( seznamFormPar )
{
      tělo (výkonný kód) metody
}

seznamFormParam = typ názevFormParametru, ...
```

Např.:

```
public void prevedNa(Ucet kam, double castka) {
    uber(castka);
    kam.pridej(castka);
}
```

Deklarace proměnné v metodě:

```
Typ jméno;
```

Ukázka deklarace třídy

```
public class Ucet {
    protected String majitel;
    protected double zustatek;

    public void pridej(double castka) {
        zustatek += castka;
    }
    public void vypisZustatek() {
        System.out.println(zustatek);
    }
    public void uber(double castka) {
        zustatek -= castka;
    }
    public void prevedNa(Ucet kam, double castka) {
        uber(castka);
        kam.pridej(castka);
    }
}
```

Použití třídy

deklarace (určení typu) proměnné

- `Typ jmeno;`
- `Ucet ucet;`
- `ucet` bude typu `Ucet`

vytvoření objektu

- `ucet = new Ucet();`
- vytvoří se objekt třídy `Ucet` a uloží se (resp. *reference na objekt*) do proměnné `ucet`

sloučení deklarace a inicializace na jeden řádek

- `Ucet ucet = new Ucet();`

Volání metod

Nad existujícími (vytvořenými) objekty můžeme volat jejich metody

- 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"
- volání lze provést, jen je-li metoda z místa volání přístupná - "viditelná"
- přístupnost regulují podobně jako u proměnných modifikátory přístupu

Ukázka použití třídy

```
public class Banka {
    public static void main(String[] args) {
        Ucet ucet = new Ucet();
        ucet.vypisZustatek();
        ucet.pridej(100.50);
        ucet.vypisZustatek();
        ucet.uber(0.50);
        ucet.vypisZustatek();
    }
}
```

Datové typy

Java striktně rozlišuje mezi hodnotami

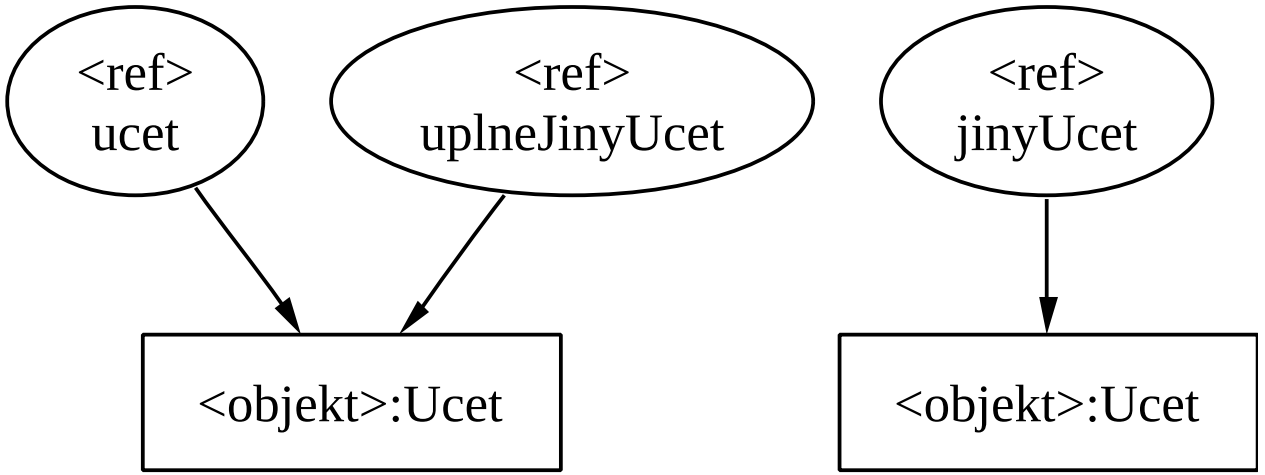
- primitivních datových typů (čísla, logické hodnoty, znaky) a
- objektových typů (řetězce a všechny uživatelem definované [tj. vlastní] typy – třídy a rozhraní)

Základní rozdíl je v práci s proměnnými:

- proměnné primitivních datových typů přímo obsahují danou hodnotu
- proměnné objektových typů obsahují pouze odkaz na příslušný objekt
- $\Rightarrow$ dvě objektové proměnné mohou nést odkaz na tentýž objekt

Proměnné objektového typu

```
public class Banka {
    public static void main(String[] args) {
        Ucet ucet = new Ucet();
        Ucet jinyUcet = new Ucet();
        Ucet uplneJinyUcet = ucet;
    }
}
```



Inicializace proměnných

- primitivní typy $\Rightarrow$ `0`
- objektové typy $\Rightarrow$ `null`

```
public class Banka {
    protected int pocetUctu;   // pocetUctu == 0
    ...
    public static void main(String[] args) {
        Ucet ucet;             // ucet == null
        ...
    }
}
```

Předávání 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.

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:

- 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`.

Přetěžování metod

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 metod

```
public void prevedNa(Ucet kam, double castka) {
      uber(castka);
      kam.pridej(castka);
}


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

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 Ucet prevedNa(Ucet kam, double castka) {
      uber(castka);
      kam.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();
     petruvUcet.pridej(100);
     ivanuvUcet.pridej(100);

     // budeme řetězit volání:
     petruvUcet.prevedNa(ivanuvUcet, 30).vypisZustatek();
     // převede 30 jednotek a vypíše zůstatek => 70

     ivanuvUcet.vypisZustatek(); // vypíše 130
}
```

Vytváření objektů

Voláním např. `new Ucet()` jsme použili:

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

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, eventuálně volat z jiného konstruktoru.

Implicitní konstruktor

Každá třídá má *implicitní* (*bezparametrický*) konstruktor

- nemá žádné parametry
- nemá žádný návratový typ!
- nemusí se deklarovat
- deklarace: `JmenoTridy() {...}`

```
public class Ucet {
    public Ucet() {
        ...
    }
}
```

Použití: `new Ucet();`

Další konstruktory

Každá třída může mít další (jiné) konstruktory než implicitní

- odlišují se parametry
- nemají návratový typ
- pokud se deklaruje alespoň jeden konstruktor, implicitní se již negeneruje!!

```
public class Ucet {
    private String majitel;
    public Ucet(String name) {
        majitel = name;
    }
}
```

Použití:

```
new Ucet("Ferda");
new Ucet();
```

$\Leftarrow$ chyba!

Další konstruktory

Pokud chceme deklarovat další konstruktory a současně používat implicitní, musíme ho také deklarovat!

```
public class Ucet {
private String majitel;
    public Ucet() {}
    public Ucet(String name) {
        majitel = name;
    }
}
```

Použití:

```
new Ucet("Ferda");
new Ucet();       ⇐ OK
```

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`

Příklad – počítání účtů

```
public class Ucet {
    protected String majitel;
    protected double zustatek = 0;
    protected static int pocet = 0;

    public Ucet(String jmeno) {
        majitel = jmeno;
        pocet++;
    }

    public static int pocetUctu() {
        return pocet;
    }
}
```

```
Ucet ucet = new Ucet("Ferda");
System.println(Ucet.pocetUctu());
```

Příklad – počítání účtů (volání konstruktoru)

```
public class Ucet {
    protected String majitel;
    protected double zustatek = 0;
    protected static int pocet = 0;

    public Ucet() {
        pocet++;
    }
    public Ucet(String jmeno) {
        this();
        majitel = jmeno;
    }

    public static int pocetUctu() {
        return pocet;
    }
}
```

Balíky

```
package seminar2.banka.ucty;
public class Ucet {
     ...
}
```

```
package seminar2.banka;
public class Banka {
     Ucet ucet = new Ucet;
     ...
}
```

```
 IJA
  |-- seminar2
            |-- banka
                    |-- Banka.java
                    |-- ucty
                           |-- Ucet.java
```

Přístup k třídám z jiných balíků

- tečková notace `seminar2.banka.ucty.Ucet`
- $\Rightarrow$ zdlouhavé, kompikované
- $\Rightarrow$ import tříd

Import tříd z balíků

- klauzule `import` *`package.třída`*
- klauzule `import` *`package.*`*
- `*` nezpřístupní třídy z podbalíků!!

```
package seminar2.banka.ucty;
public class Ucet {
     ...
}
```

```
package seminar2.banka;
import seminar2.banka.ucty.Ucet;
public class Banka {
     Ucet ucet = new Ucet;
     ...
}
```

Import tříd z balíků – překlad

```
IJA
 |-- seminar2
          |-- banka
                   |-- Banka.java
                   |-- ucty
                          |-- Ucet.java
```

Př.: jsme v adresáři `IJA`

```
javac -classpath .  seminar2/banka/ucty/Ucet.java
javac -classpath .  seminar2/banka/Banka.java

java -classpath .  seminar2.banka.Banka
```

Import tříd z balíků

- balík `java.lang` je vždy importován automaticky
- třída `java.lang.System`

Modifikátory přístupu

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

- Přístupem se rozumí jakékoli použití dané třídy, prvku...
- Omezení přístupu je kontrolováno hned při překladu $\Rightarrow$ není-li přístup povolen, nelze program ani přeložit.
- Tímto způsobem lze regulovat přístup staticky, mezi celými třídami, nikoli pro jednotlivé objekty

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

- `public` = veřejný
- `protected` = chráněný
- modifikátor neuveden = říká se lokální v balíku nebo chráněný v balíku nebo "přátelský"
- `private` = soukromý

Pro třídy:

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

Typy omezení přístupu

Pro vlastnosti tříd = proměnné/metody:

- veřejné (`public`)
- chráněné (`protected`)
  - přístupné jen ze tříd stejného balíku a z podtříd
- neveřejné (lokální v balíku)
  - přístupné jen ze tříd stejného balíku, už ale ne z podtříd, jsou-li v jiném balíku (nedoporučuje se)
- soukromé (`private`)
  - přístupné jen v rámci třídy – používá se častěji pro proměnné než metody
  - zneviditelníme i případným podtřídám

Typy omezení přístupu

```
public class Ucet {
    protected String majitel;
    protected double zustatek;

    public void pridej(double castka) {
        zustatek += castka;
    }
    public void vypisZustatek() {
        System.out.println(zustatek);
    }
    public void uber(double castka) {
        zustatek -= castka;
    }
}
```

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.

Primitivní datové typy

- Proměnné těchto typů nesou atomické, dále nestrukturované hodnoty
- Deklarace způsobí
  - rezervování příslušného paměťového prostoru
  - zpřístupnění (pojmenování) tohoto prostoru identifikátorem proměnné

Primitivní datové typy

Typ logických hodnot – `boolean`

- přípustné hodnoty jsou false a true
- na rozdíl od Pascalu na nich není definováno uspořádání

Typ `void`

- není v pravém slova smyslu datovým typem, nemá žádné hodnoty
- označuje "prázdný" typ pro sdělení, že určitá metoda nevrací žádný výsledek

Primitivní datové typy

Čísla s pohyblivou řádovou čárkou

- `float`
  - 32 bitů
- `double`
  - 64 bitů
- zápis literálů
  - ```
    float f = -.777f, g = 0.123f, h = -4e6f,
    i = 1.2E-15f;
    ```
  - ```
    double f = -.777, g = 0.123, h = -4e6,
    i = 1.2E-15;
    ```

Primitivní datové typy

Integrální typy – celočíselné

- v Javě jsou celá čísla vždy interpretována jako znaménková
- `int`
  - 32 bitů ($-2\ 147\ 483\ 648$ .. $2\ 147\ 483\ 647$)
  - základní celočíselný typ
- `long`
  - 64 bitů (cca $+/-\ 9 \cdot 10^{18}$)
- `short`
  - 16 bitů (-32768 .. 32767)
- `byte`
  - 8 bitů (-128 .. 127)

Primitivní datové typy

Integrální typy – `char`

- char představuje jeden 16bitový znak v kódování UNICODE
- konstanty typu char zapisujeme
  - v apostrofech: `'a'`, `'ř'`
  - pomocí escape-sekvencí: $\backslash n$ (konec řádku) $\backslash t$ (tabulátor)
  - hexadecimálně: $\backslash u0040$ (totéž, co `'a'`)
  - oktalově: $\backslash 127$
- *Pozor na kódové stránky při překladu/spouštění – dochází k překódování textu! (komentář, znak, řetězec, identifikátor)*