Seminář Java

Programování v Javě – I

Radek Kočí

Fakulta informačních technologií VUT

Únor 2010

Téma přednášky

- Datové typy
- Deklarace třídy
- Modifikátory přístupu
- Struktura aplikace v Javě

Datové typy

Java striktně rozlišuje mezi hodnotami

- primitivních datových typů
  - čísla
  - logické hodnoty
  - znaky
- objektových typů
  - řetězce
  - uživatelem definované 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

Primitivní datové typy

Charakteristika

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

Primitivní datové typy

Typ `boolean`

- logická hodnota, 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: \n (konec řádku) \t (tabulátor)
  - hexadecimálně: \u0040 (totéž, co `'a'`)
  - oktalově: \127
- *Pozor na kódové stránky při překladu/spouštění – dochází k překódování textu! (komentář, znak, řetězec, identifikátor)*

```
javac -encoding ISO8859-2 ...
```

Proměnné objektového typu

Charakteristika

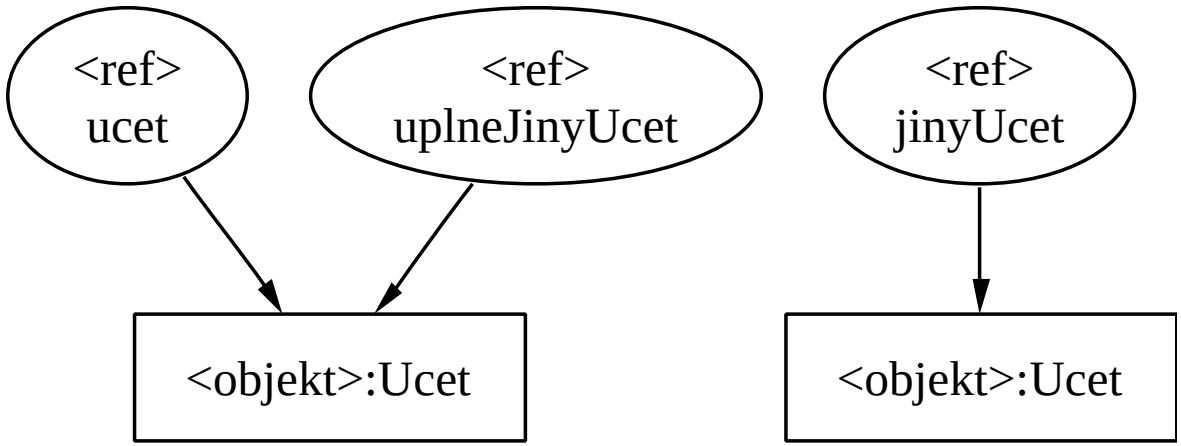
- Proměnné těchto typů reprezentují reference na objekty
- Deklarace způsobí
  - rezervování paměťového prostoru pro *referenci*!
  - vlastní objekt (instance třídy) nevzniká!

Vytvoření instance

- operátor `new`
- rezervuje paměťový prostor pro objekt (instanci dané třídy)

Proměnné objektového typu

```
Ucet ucet = new Ucet();
Ucet jinyUcet = new Ucet();
Ucet uplneJinyUcet = ucet;
```



Deklarace třídy – Příklad

Chceme modelovat následující skutečnost

- máme dopravce vlastnící dopravní prostředky
- dopravce má zaměstnance, kteří mohou řídit dopravní prostředky
- každý prostředek může být řízen některým ze zaměstnanců

Deklarace třídy – Příklad

```
public class Car {
    protected Driver drivenBy;
    public void drivenBy(Driver d) {
        drivenBy = d;
    }
}

public class Driver {
    protected String name;
    protected Car driveCar;
    public void driveCar(Car c) {
        driveCar = c;
        driveCar.drivenBy(this);
    }
}
```

Deklarace třídy

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

Modifikátory přístupu

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

Deklarace proměnné objektu

Deklarace proměnné objektu má tvar:

```
modifikátory Typ jméno;
```

např.:

```
protected int capacity;
```

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

Deklarace proměnné objektu

Inicializace proměnné objektu (primitivní datový typ)

- **`int capacity;`**
- *`capacity`* `== 0`

Inicializace proměnné objektu (objektový typ)

- **`Driver drivenBy;`**
- *`drivenBy`* `== null`

Deklarace metody

```
modifikátory typ název ( seznamFormPar )
{
      tělo (výkonný kód) metody
}
```

seznamFormParam = typ názevFormParametru, ...

Např.:

```
public void run(String name, Car car) {
   collection.getDriver(name).driveCar(car);
}
```

Deklarace proměnné v metodě

Deklarace proměnné v metodě

- deklarace bez modifikátorů
- nemá implicitní inicializaci

```
public void run(String name, Car car) {
    Driver d = collection.getDriver(name);
    d.driveCar(car);
}
```

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 "tečkovou notací"
- volání lze provést, jen je-li metoda z místa volání přístupná
- přístupnost regulují modifikátory přístupu

Volání metod

```
...
Car car = new Car();
Driver driver = new Driver();

driver.driveCar(car);
...
```

Předávání parametrů metodám

Hodnoty primitivních typů

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

Hodnoty objektových 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 se buď předává kopie hodnoty primitivního typu, nebo kopie hodnoty odkazu (reference) na objekt.

Návrat z metody

Kód metody skončí jakmile

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

Metoda může při návratu vrátit hodnotu (chovat se jako funkce)

- vrácenou hodnotu musíme uvést za příkazem `return`
- typ vrácené hodnoty musíme v hlavičce metody deklarovat
- nevrací-li metoda nic, pak musíme namísto typu vracené hodnoty psát `void` (v tomto případě se `return` nemusí uvádět)

Ukončení metody způsobí předání řízení

- zpět volající metodě + předání případné hodnoty

Proměnné a metody třídy – statické

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

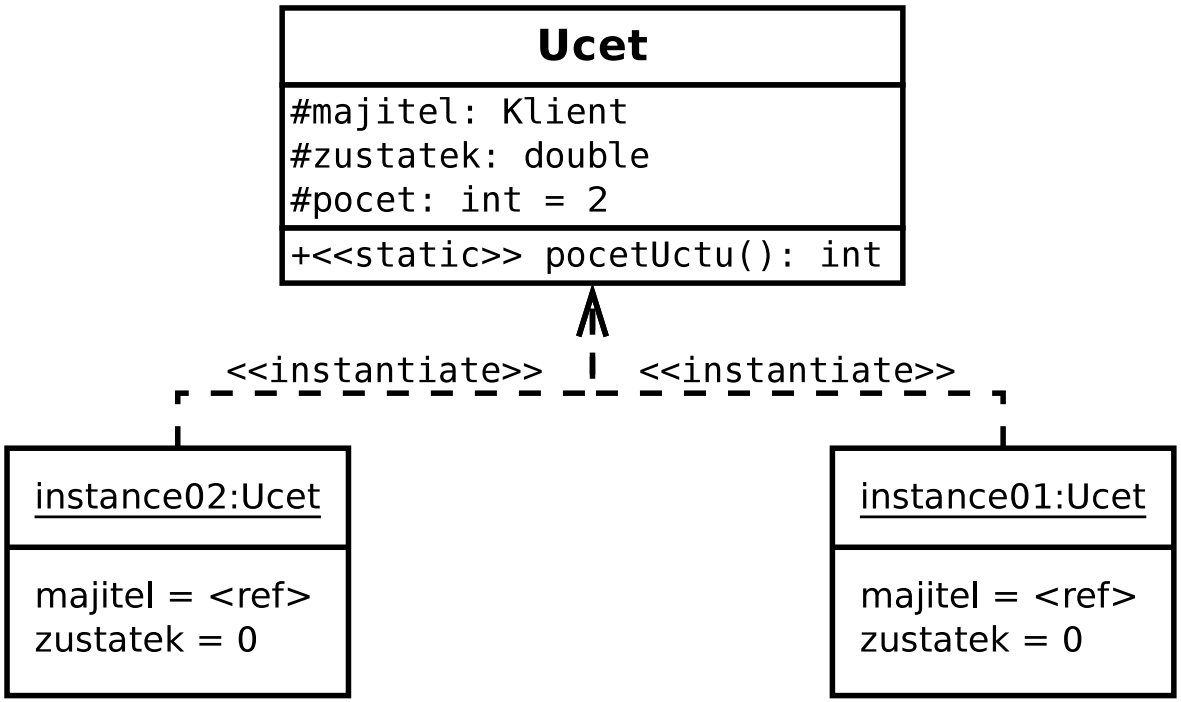
- Lze deklarovat také metody a proměnné patřící celé třídě, tj. skupině všech objektů daného typu.
- Takové metody a proměnné nazýváme statické a označujeme v deklaraci modifikátorem `static`

Příklad – počítání účtů

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

    public Ucet(Klient k) {
        majitel = k;
        pocet++;
    }
    public static int pocetUctu() {
        return pocet;
    }
}
```

Příklad – počítání účtů




```
Ucet instance01 = new Ucet(k);
Ucet instance02 = new Ucet(k);
int pocet = instance01.pocetUctu();

pocet = Ucet.pocetUctu();   <= pristup pres tridu
```

Základní životní cyklus programu v Javě

Programování v Javě spočívá ve vytváření tříd, neexistují metody a atributy deklarované mimo třídy.

Základní životní cyklus programu v Javě

- aplikace sestává z alespoň jedné třídy
- zdrojový kód každé veřejně přístupné třídy (`public`) je umístěn ve zvláštním souboru
  - `NazevTridy.java` (povinná přípona!)
- každá přeložená třída má svůj soubor s bytecode
  - `NazevTridy.class` (povinná přípona!)
- třídy jsou organizovány do balíků (packages)
- u běžné "desktopové" aplikace představuje vstupní bod do programu třída obsahující metodu `main`

- Java je *case sensitive*! (`ucet` x `Ucet`)
- API: `http://java.sun.com/reference/api/`

Základní životní cyklus programu v Javě

Ukázka aplikace

- třída Pozdrav
- je umístěná v souboru Pozdrav.java
- je umístěna v balíku ija1

```
package ija1;
public class Pozdrav {
  // Program spouštíme aktivací funkce "main"
  public static void main(String[] args) {
    System.out.println("Ahoj!");
  }
}
```

Co znamená spustit program?

Spuštění javového programu odpovídá spuštění metody *main* jedné ze tříd tvořících program

Aplikace může mít parametry:

- podobně jako např. v jazyku C
- jsou typu `String` (řetězec)
- předávají se při spuštění z příkazového řádku do pole `String[] args` (argument metody *main*)

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

- nevrací žádnou hodnotu – návratový typ je vždy(!) void
- její hlavička musí vypadat vždy přesně tak, jako ve výše uvedeném příkladu, jinak nebude spuštěna!

Návrat z metody

Kód metody skončí jakmile

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

Ukončení metody `main` způsobí předání řízení systému (JVM).

Základní životní cyklus programu v Javě

Vytvoření zdrojového textu

- libovolný editor $\Rightarrow$ `Pokus.java`

Překlad

- `javac Pokus.java`
- název souboru se udává včetně přípony `.java`
- vznikne soubor `Pokus.class`

Spuštění

- `java Pokus`
- udává se název třídy (tj. bez přípony `.class`)

Spuštění se nepovede, viz organizace tříd do balíků ...

Organizace tříd do balíků

- třídy jsou členěny do balíků (package)
- balíky vytvářejí stromovou strukturu
- organizaci balíků odpovídá organizace adresářů a umístění zdrojového souboru do příslušného adresáře
- může existovat více stromů

- Třída je plně kvalifikovaná svým názvem a balíkem!
- `ija1.Pozdrav`

Organizace tříd do balíků

```
package ija1;
```

```
$HOME
  |-- IJA
       |-- ija1
            |-- Pozdrav.java
            |-- Pozdrav.class
```

Překlad a spuštění

1. jsme v adresáři `$HOME/IJA`
2. spustíme překlad `javac ija1/Pozdrav.java`
3. spustíme aplikaci `java ija1.Pozdrav`

Organizace tříd do balíků

```
package ija1;
```

```
$HOME
  |-- IJA
       |-- ija1
            |-- Pozdrav.java
            |-- Pozdrav.class
```

Překlad a spuštění

1. jsme v adresáři `$HOME/IJA`
2. spustíme překlad `javac ija1/Pozdrav.java`
3. spustíme aplikaci `java ija1.Pozdrav`

Organizace tříd do balíků

```
$HOME
    |-- java
          |-- distribution
          |-- project
                |-- ija1
                |-- ija2
          |-- docs
    |-- sun
          |-- distribution
          |-- examples
                |-- ija3
          |-- docs
```

Kořenový adresář:
`$HOME/java/project`
`$HOME/sun/examples`

Organizace tříd do balíků

Nastavení cest pro balíky

- balíky (kořeny stromů) mohou být umístěny v různých adresářích
- je možné nastavit cesty do těchto adresářů
- v těchto adresářích se pak hledají balíky a třídy

Systémová proměnná `CLASSPATH`

```
export
CLASSPATH="$CLASSPATH:$HOME/java/project:..."
```

Parametr `-classpath`

```
javac -classpath "$HOME/java/project:..." ...
java -classpath "$HOME/java/project:..." ...
```

Ukázka aplikace

```
package ija1;
public class Pozdrav { ...  }
```

```
$HOME
  |-- IJA
        |-- ija1
              |-- Pozdrav.java
```

Překlad (lze z libovolného adresáře):

```
javac -classpath "$HOME/IJA"
                 $HOME/IJA/ija1/Pozdrav.java
```

Spuštění (lze z libovolného adresáře):

```
java -classpath "$HOME/IJA" ija1.Pozdrav
```

Ukázka deklarace třídy

```
package ija1.cars;

public class Car {
    protected int weight;
    protected int capacity;
    protected Driver drivenBy;
    public void drivenBy(Driver d) {
        drivenBy = d;
    }
    public int getCapacity() {
        return capacity;
    }
}
```

`ija1.cars.Car car = new ija1.cars.Car();`

Přístup k třídám z jiných balíků

- tečková notace `ija1.cars.Car`
- $\Rightarrow$ zdlouhavé, komplikované
- $\Rightarrow$ import tříd

```
package homework1;
public class Homework {
   ija1.cars.Car c;
   c = new ija1.cars.Car();
   ...
}
```

Import tříd z balíků

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

```
package homework1;
import ija1.cars.Car;
//import ija1.cars.*;

public class Homework {
    Car c;
    c = new Car();
    ...
}
```

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
- takto lze regulovat přístup staticky, mezi celými třídami, nikoli pro jednotlivé objekty

Granularita omezení přístupu

- přístup je v Javě regulován jednotlivě po prvcích
- omezení přístupu je určeno uvedením jednoho z modifikátoru přístupu (access modifier) nebo neuvedením žádného

Modifikátory přístupu

Typy omezení přístupu

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

Pro třídy:

- veřejné (*public*)
  - přístup k třídě není omezen
- neveřejné (*lokální v balíku*)
  - k třídě může přistupovat libovolná třída ze stejného balíku

Modifikátory přístupu

Pro vlastnosti tříd (proměnné/metody):

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

Příklad

```
public class Car {
    protected int weight;
    protected int capacity;
    protected Driver drivenBy;

    public void drivenBy(Driver d) {
        drivenBy = d;
    }
    public int getWeight() {
        return weight;
    }
    public int getCapacity() {
        return capacity;
    }
}
```

Speciální modifikátor `final`

- Deklaruje konečný (neměnný) stav
- Třídy
  - `public final class Ucet { ... }`
  - od této třídy nelze "dědit" (vytvářet její potomky)
- Metody
  - `public final void print() { ... }`
  - tato metoda nemůže být "překryta" (overloaded) v odvozených třídách (potomci)
- Proměnné
  - `protected final int i = 10;`
  - `protected final String s = "řetězec";`
  - `protected final Banka b = new Banka();`
  - obsah proměnné je neměnný
  - konstanta

Příklad

```
public final class System {
    public static final PrintStream out;
    public static final InputStream in;
    ...

    public static long currentTimeMillis();
    ...
}
```

```
public static void main(String[] argv) {
    ...
}
```