Seminář Java

Speciální třídy, výčtový typ

Radek Kočí

Fakulta informačních technologií VUT

Únor 2008

Téma přednášky

- Abstraktní třídy
- Vnořené třídy
- Anonymní třídy
- Výčtový typ (enums)
- Proměnný počet parametrů (varargs)
- Statický import

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

Abstraktní třída

Abstraktní třída

- třída, která danou specifikaci implementuje jen částečně
- nemůže mít instance
- klíčové slovo **`abstract`**

Abstraktní třída = částečná implementace
Třída = úplná implementace

Abstraktní třída – Příklad

```
public class Vehicle {
    ...
    public int price(int km) { ??? }
}

public class Car extends Vehicle {
    protected int price(int km) {
        // podle osobniho auta ...
    }
}

public class Bus extends Vehicle {
    protected int price(int km) {
        // podle autobusu ...
    }
}
```

Abstraktní třída – Příklad

```
public abstract class Vehicle {
    ...
    public Vehicle(int weight, int capacity) {
        this.weight = weight;
        this.capacity = capacity;
    }
    protected abstract int price(int km);
}

public class Car extends Vehicle {
    public Car() {
        super(2000, 5);
    }
    protected int price(int km) { ... }
}
```

Top-level classes

Třídy nejvyšší úrovně (top-level classes)

- ”normální třídy” – jsou přímými členy nějakého balíku

```
package balik;

public class TopLevel1 {
    ...
}
```

Vnořené třídy (inner classes)

Lokální třídy

- vnořené v jiné třídě (na úrovni lokálních proměnných)
- uváděné uvnitř bloku (platné pouze v uvedeném bloku)
- nesmí být *public*, *private* a *protected*

Vnořené třídy (inner classes)

```
public class TopLevel1 {
  private String text = "interni promenna";
  public void test() {
    {
      class A {
        public A() {
          System.out.println(text);
        }
      }
      A a = new A();
    }
    // tady uz neni trida A dostupna
  }
}
```

Vnořené třídy (inner classes)

Členské třídy (member classes)

- vnořené v jiné třídě (na úrovni vlastností objektu)

```
public class TopLevel1 {
  private String text = "interni promenna";
  class Inner {
    public Inner() {
      System.out.println("Inner:  " + text);
    }
  }
  public m() {
    Inner i = new Inner();
    ...
  }
}
```

Vnořené třídy (inner classes)

Vnořené top-level třídy

- členské třídy s modifikátorem **`static`**
- vnořená rozhraní
- mohou být *public*, *private* a *protected*

Vnořené třídy (inner classes)

```
public class TopLevel1 {
  private String text = "interni promenna";
  static public class TopLevel2 {
    public TopLevel2() {
      TopLevel1 t = new TopLevel1();
      System.out.println(t.text);
    }
  }
  interface Cool {
    ...
  }
}
```

Vnořené třídy (inner classes)

Vnořené top-level třídy

- používá se k seskupení souvisejících tříd bez nutnosti vytvářet nový balík
- přístup k vnořeným top-level třídám (rozhraním)
  `new TopLevel1.TopLevel2();`

Anonymní třídy

Anonymní třídy (anonymous classes)

- zvláštní případ vnořené třídy

```
new Typ ( parametry ) {
    tělo anonymní třídy
}
```

- Typ představuje
  - jméno konstruktoru rodičovské(!) třídy, od které je anonymní třída odvozena (následují jeho parametry)
  - jméno rozhraní – anonymní třída jako jediná může přímo instanciovat rozhraní (zde se parametry neuvádějí)

Anonymní třídy

```
class NejakaTrida {
    Runnable r = new Runnable() {
        public void run() {
            // ...
        }
    }
}
```

Enums

Reprezentace výčtového typu (Java $<$ 5.0)

- int Enum pattern

```
public static final int SEASON_WINTER = 0;
public static final int SEASON_SPRING = 1;
public static final int SEASON_SUMMER = 2;
public static final int SEASON_FALL = 3;
```

- není typově bezpečný
- hodnoty nejsou informativní (vždy int; ”jiné” názvy, stejná hodnota)

Enums

Výčtový typ (Java $>=$ 5.0)

- typově bezpečný
- `enum Season {WINTER, SPRING, SUMMER, FALL}`
- plnohodnotná třída!
- rozšiřuje třídu `java.lang.Enum`

Enums

```
class EnumsDemo {
    public enum Season { WINTER, SPRING, SUMMER,
                         FALL };
    private Season season;

    public static void main(String[] args) {
        new EnumsDemo();
    }
    public EnumsDemo() {
        season = Season.WINTER;
        System.out.println(season);
        for (Season s : Season.values())
            System.out.println("Value of Season: "
                               + s);
    }
}
```

Enums

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6);

    private final double mass;   // in kilograms
    private final double radius; // in meters

    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
```

Enums

```
    public static final double G = 6.67300E-11;

    double surfaceGravity() {
        return G * mass / (radius * radius);
    }

    double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Enums

```
public static void main(String[] args) {
  double earthWeight = Double.parseDouble(args[0]);
  double mass =
     earthWeight/Planet.EARTH.surfaceGravity();

  for (Planet p : Planet.values())
    System.out.printf("Your weight on %s is %f%n",
                      p, p.surfaceWeight(mass));
}
```

---

http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html

Varargs

Varargs

- metoda s proměnným počtem parametrů
- pole objektů třídy **Object**

```
Object[] arguments = {
    new Integer(7),
    new Date(),
    "a disturbance in the Force"
};

String result = MessageFormat.format(
    "At {1,time} on {1,date}, there was {2} " +
    "on planet {0,number,integer}.", arguments);
```

Varargs

Varargs (Java $>=$ 5.0)

- vlastnost skrývající nutnost obalení polem
- poslední argument: `Object... name`

Varargs

```
class Args {

    public static void main(String[] args) {
        Args a = new Args();
        a.print("Hi ", "hou ", "ha");
    }

    public void print(String s, Object... args) {
        String str = "";
        System.out.println(args.length);
        for (Object o : args)
            str += (String) o;
        System.out.println(s + str);
    }
}
```

Static import

Přístup ke statickým členům

- `double r = Math.cos(Math.PI * theta);`
- Využití statického importu

```
 import static java.lang.Math.PI;
//import static java.lang.Math.*;
...

double r = cos(PI * theta);
```

- používat velice opatrně! (kolize identifikátorů, těžko čitelný kód, ...)