Seminář Java

Programování v Javě – III

Radek Kočí

Fakulta informačních technologií VUT

Březen 2010

Téma přednášky

- Pole
- Dokumentační komentáře
- Java Archive – JAR
- Apache Ant
- Ladění programu

Pole

Pole

- 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 – *kontejnery* (bude později ...)

Pole – I

Před použitím je nutné pole

- deklarovat
- vytvořit
- inicializovat (naplnit)

Syntaxe deklarace

- **`typ [] identifikator`**
- na rozdíl od C/C++ nikdy neuvádíme při deklaraci počet prvků pole – ten je podstatný až při vytvoření objektu pole

Pole – II

Vytvoření pole

- jako u jiného objektu – voláním konstruktoru:
  - **`nazevPole = new typ[velikost];`**
  - **`int[] pole = new int[10];`**
- nebo inicializací při deklaraci:
  - **`int[] nazevPole = { 1, 2, 3 };`**

Syntaxe přístupu k prvkům

- **`pole[i]`**
- **`pole[i] = 20;`**
- **`int j = pole[2];`**

Pole – III

```
Ucet [] ucty;                    // deklarace pole
ucty = new Ucet[5];              // vytvoření pole

// vytvoření objektu
// a inicializace 1.  prvku pole
ucty[0] = new Ucet("Franta");

ucty[0].vypisInfo();  // přístup k prvku pole
```

- V poli `ucty` je naplněn 1. prvek odkazem na objekt
- Ostatní prvky zůstaly naplněny prázdnými odkazy `null`.

Pole – IV

Co když vynecháme vytvoření pole?

```
Ucet [] ucty;
ucty[0] = new Ucet("Franta");
   // chyba, pole neexistuje
```

Co když vynecháme inicializaci pole?

```
Ucet [] ucty;
ucty = new Ucet[5];
ucty[0].vypisInfo();
   // chyba, prvek neexistuje
```

Kopírování pole

Přiřazení proměnné objektového typu (a tedy i polí) vede pouze k duplikaci odkazu, nikoli celého odkazovaného objektu.

```
Ucet [] ucty = new Ucet[5];
Ucet [] ucty2;
ucty2 = ucty;
```

Proměnná `ucty2` obsahuje odkaz na stejné pole jako `ucty`.

Kopírování pole

```
Ucet [] ucty2 = new Ucet[5];
System.arraycopy(ucty, 0, ucty2, 0,
                 ucty.length);
```

- Proměnná `ucty2` obsahuje kopii původního pole.
- Také `arraycopy` však do cílového pole zduplikuje jen odkazy na objekty, nevytvoří kopie objektů!

Výpis argumentů programu

```
public class Pole {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++)
            System.out.println(args[i]);
    }
}
```

Komentáře

Základní typy komentářů (podobně jako např. v C/C++)

- *řádkové* od značky `//` do konce řádku
- *blokové* (na libovolném počtu řádků) začínají `/*` pak je text komentáře, končí `*/`
- *dokumentační* (na libovolném počtu řádků) od značky `/**` po značku `*/` Každý další řádek může začínat mezerami či *, hvězdička se v komentáři neprojeví.

Komentáře

```
// řádkový komentář

/*
   blokový
   (víceřádkový) komentář
*/

/**
   dokumentační
   (víceřádkový) komentář
*/
```

Generování dokumentace

Dokumentace

- je generována nástrojem `javadoc`
  - z dokumentačních komentářů
  - a ze samotného zdrojového textu
- *je tedy možné dokumentovat (základním způsobem) i program bez vložených komentářů!*
- má standardně podobu HTML stránek (s rámy i bez)
- chování `javadoc` můžeme změnit volbami (options) při spuštění

Dokumentační komentáře uvádíme:

- před hlavičkou třídy (komentuje třídu jako celek)
- před hlavičkou metody nebo proměnné (komentuje příslušnou metodu nebo proměnnou)

Značky pro dokumentační komentáře

Nástroj `javadoc` můžeme podrobněji instruovat pomocí značek vkládaných do dokumentačních komentářů, např.:

@author	specifikuje autora API/programu
@version	označuje verzi API, např. ”1.4.2”
@deprecated	informuje, že prvek je zavrhovaný
@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)

Ukázka použití dokumentačních komentářů

```
package ija1.ucty;

/**
 * Trida ucet
 * @author R. Koci
**/
public class Ucet {
    /** Majitel uctu */
    protected String majitel;
    protected double zustatek;

    public void pridej(double castka) {
        zustatek += castka;
    }
    ...
}
```

Java Archive (JAR)

- platformově nezávislý
- archiv (zip) obsahující
  - hierarchii balíků a class soubory
  - jakékoliv jiné soubory (obrázky pro aplety apod.)
  - speciální adresář `META-INF`

`META-INF`

- obsah je interpretován JVM
- konfigurace aplikace
- konfigurace rozšíření
- konfigurace zavaděčů tříd a služeb

Zkompilování zdrojových textů

```
-- src
    |---- xml
            |---- XMLDemo.java
-- dest
-- dom4j-1.5.2.jar
```

```
javac -classpath "src:dom4j-1.5.2.jar"
              -d dest src/xml/XMLDemo.java
```

Zkompilovani zdrojovych textu

```
-- src
    |---- xml
            |---- XMLDemo.java
-- dest
    |---- xml
            |---- XMLDemo.class
-- dom4j-1.5.2.jar
```

Java Archive (JAR)

```
-- dest
    |---- xml
            |---- XMLDemo.class
-- dom4j-1.5.2.jar
```

```
jar -cvf xml.jar -C dest xml
```

```
MANIFEST.MF:

Manifest-Version: 1.0
Created-By: 1.5.0_05 (Sun Microsystems Inc.)
```

Java Archive (JAR)

```
-- dest
    |---- xml
            |---- XMLDemo.class
-- dom4j-1.5.2.jar
```

```
jar -cvfm xml.jar mymanifest.mf -C dest xml
```

```
MANIFEST.MF:
Manifest-Version: 1.0
Class-Path: dom4j-1.5.2.jar
Created-By: 1.5.0_05-b05 (Sun Microsystems Inc.)
Ant-Version: Apache Ant 1.6.2
Main-Class: xml.XMLDemo
```

Java Archive (JAR)

Spuštění JAR souboru

```
java -jar xml.jar
```

---

http://java.sun.com/j2se/1.5.0/docs/guide/jar/

Apache Ant

Apache Ant

- nástroj pro sestavování aplikací
- v principu podobný nástroji `make`
- sestavovací soubory založeny na XML formátu

Sestavovací soubor (buildfile)

- projekt (project)
- cíle (targets)
- úlohy (tasks)
- závislosti

`http://ant.apache.org`

Apache Ant

Soubor `build.xml`

```
<project>
  <target name="clean">
    <delete dir="build"/>
  </target>

  <target name="compile">
    <mkdir dir="build/classes"/>
      <javac srcdir="src" destdir="build/classes"/>
  </target>
```

```
    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/xml.jar"
                        basedir="build/classes">
            <manifest>
                <attribute name="Main-Class"
                        value="xml.XMLDemo"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/xml.jar" fork="true"/>
    </target>
</project>
```

Apache Ant

```
ant compile
ant jar
ant run
```

Ladění programu

Pro ladění programů v Javě lze využít

- kontrolní tisky: `System.err.println(...)`
- řádkový debugger `jdb`
- integrovaný debugger v IDE
- speciální nástroje na záznam běhu balíků

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

Ladění programu

- standardní klíčové slovo (od JDK1.4) `assert`
  - `assert booleovský_výraz`
- testovací nástroje typu JUnit (a varianty – `HttpUnit`, ...)
  - metoda `assertEquals()`
  - metoda `assertTrue()`
  - ...
  - `http://junit.org/`
- pokročilé nástroje na běhovou kontrolu platnosti invariantů, vstupních, výstupních a dalších podmínek
  - např. jass (Java with ASSertions),
  - `http://csd.informatik.uni-oldenburg.de/~jass/`

Ladění programu – assert

```
public class AssertDemo {
    public static void main(String args[]) {
        int x = 10;
        boolean enabled = false;

        assert enabled = true;

        System.out.println("Assertions are " +
            (enabled ? "enabled" : "disabled");

        assert x < 0 : "x is not < 0";
    }
}
```

Ladění programu – assert

- spustit s volbou `-ea` (`-enableassertions`)
- dojde-li za běhu programu k porušení podmínky stanovené za `assert`, vznikne běhová chyba (`AssertionError`) a program skončí

Ladění programu – JUnit

Instalace JUnit

- stáhnout si distribuci testovacího prostředí (stačí binární) `http://junit.org`
- nainstalovat (rozbalit do adresáře) → archiv jar

Java Archive (JAR)

- archiv (zip) obsahující
  - hierarchii balíků a class soubory
  - jakékoliv jiné soubory (obrázky pro aplety apod.)
- použití
  - `javac -cp junit.jar ...`

Ladění programu – JUnit

Postup (starší verze)

- napsat testovací třídu (třídy) – obvykle rozšiřují (dědí) třídu `junit.framework.TestCase`
- testovací třída obsahuje metody
  - metodu pro nastavení testu – `setUp()`
  - testovací metody – `test`*`Neco`*`()`
  - úklidovou metodu – `tearDown()`
- testovací třídu spustit v textovém nebo grafickém prostředí
  - `junit.textui.TestRunner`
  - `junit.swingui.TestRunner`
- testování zobrazí, které testovací metody případně selhaly

Ukázka (starší verze)

```
public class JUnitDemo extends TestCase {
   Zlomek x, y, z;

   public void setUp() {
      x = new Zlomek(2,3);
      y = new Zlomek(4,6);
      z = new Zlomek(4,3);
   }
   public void testRovna() {
      assertEquals("2/3 = 4/6.", x, y);
   }
   public void testSoucet() {
      Zlomek z = x.plus(y);
      assertEquals("2/3 + 4/6 = 4/3.", z, soucet);
   }
}
```

Ladění programu – JUnit

Využití anotací (annotation)

- anotace definují dodatečné informace a data o programu bez přímého vlivu na program
- využití anotací
  - při kompilaci – detekce chyb, možnost generování dalšího kódu, ...
  - za běhu – nástroje a knihovny mohou na základě anotace přizpůsobit sémantiku

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, ...)

Ladění programu – JUnit

Ukázka využití anotací (annotation)

```
import org.junit.*;
import static org.junit.Assert.*;

public class TestHW {
    @Test public void xxx() {
        ...
        assertTrue(x>10);
    }
```

```
java org.junit.runner.JUnitCore homework1.TestHW
```