# Seminář Java

## Nástroje

Radek Kočí

Fakulta informačních technologií VUT

Březen 2012

# Téma přednášky

- javadoc
- java archiv (JAR)
- ant
- ladění programu (JUnit)

# 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.

# 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

# Generování dokumentace

Dokumentační komentáře uvádíme:

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

Dokumentační komentář

- je typicky přes více řádků
- úvodní mezery a hvězdička na dalších řádcích jsou ignorovány
- bez hvězdičky mezery nejsou ignorovány
- lze vkládat html značky

# Generování dokumentace

Dvě sekce v dokumentačních komentářích

- nejdříve hlavní popis
- poté sekce s tagy; začíná prvním tagem (`@nazev_tagu`)
- pořadí sekcí nelze prohodit

Hlavní popis

- první věta komentáře je shrnutí (končí první tečkou nebo prvním tagem)
- shrnutí se zobrazí v přehledu členů třídy nebo v krátkém popisu třídy

# Ukázka použití dokumentačních komentářů

```
package ija1.ucty;

/**
 * Hlavní popis třídy účet.
 * Popis pokračuje.
 * @author R. Koci
**/
public class Ucet {
    /** Majitel uctu */
    protected String majitel;
    /** Metoda ...  */
    public void pridej(double castka) {

        zustatek += castka;
    }
    ...
}
```

# Druhy tagů

Blokové

- `@tag`
- samostatné, mohou stát jen na začátku řádku (úvodní mezery a hvězdičky ignorovány)
- znak @ uvedený kdekoliv jinde je chápán jako normální znak

In-line

- `{@tag}`
- kdekoliv v textu včetně hlavního popisu

# Generování dokumentace

Dědění komentářů

- pokud není komentář definován, dědí se od nadřazené třídy
  - u předefinovaných metod
  - u implementovaných metod z interfaců

Dokumentační komentář k balíku

- soubor `package.html` umístěný ve stejném adresáři
- do komentáře se umístí vše mezi tagy `<body>` a `</body>`
- první věta je krátký popis balíku

# 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) |
| {@link} | podobné jako @see, in-line |

# Generování dokumentace

## Spuštění

- `javadoc [parametry] [baliky]`
  `[zdrojove_soubory]`
  `[-subpackages pkg1:pkg2:...]`

## Přepínače

- `-public` : dokumentace bude obsahovat pouze public elementy
- `-protected` : dokumentace bude obsahovat public a protected elementy (implicitní chování)
- `-package` : dokumentace bude obsahovat public, protected a elementy bez označení
- `-private` : dokumentace bude obsahovat všechny elementy

# Generování dokumentace

## Přepínače

- `-source 1.4` : pokud se používají assertions
- `-encoding kodovani` : kódování zdrojových souborů
- `-d cesta` : kam vygenerovat dokumentaci
- `-version` : zahrnout tag @version
- `-author` : zahrnout tag @author
- `-windowtitle text`
- `-doctitle text`
- `-header text` : bude na začátku každé stránky
- `-footer text` : bude na konci každé stránky

## Více na

`http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html`

## 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 (`http://ant.apache.org`)

- nástroj pro sestavování aplikací
- v principu podobný nástroji **`make`**
- sestavovací soubory založeny na XML formátu
- implicitní sestavovací soubor **`build.xml`**

Sestavovací soubor (buildfile) obsahuje

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

Každý buildfile jeden projekt a alespoň jeden cíl

# Apache Ant – project

atributy

- `name` : jméno projektu
- `default` : implicitní target, který se bude provádět, pokud nebude žádný explicitně zadán
- `basedir` : adresář, od něhož budou všechny cesty v souboru odvozeny

volitelný element `<description>`

- popis projektu

```
<project name="Projekt" default="compile"
basedir=".">
<description>Dlouhy popis projektu</description>
```

# Apache Ant – target

popis

- posloupnost úloh, které se mají provést
- může záviset na jiných cílech

atributy

- `name` : jméno cíle
- `depends` : seznam cílů, na kterých závisí
- `description` : krátký popis
- `if` : jméno property, která musí být nastavena
- `unless` : jméno property, která nesmí být nastavena

```
<target name="compile" depends="init"
description="Prelozi aplikaci">
....
</target>
```

# Apache Ant – property

popis

- jméno a hodnota (u jména rozlišování velikosti písmen)
- získání obsahu property - `${property}`

vestavěné property

- `basedir`
- `ant.file`
- `ant.version`
- `ant.project.name`
- `ant.java.version`
- systémové properties Javy

vlastní property

- `<property name="jmeno" .... />`

# Apache Ant – task

popis

- kód, který může být vykonán
- různý počet parametrů, podle druhu
- vestavěné
- volitelné
- vlastní

```
<jmeno atr1="hodnota" atr2="hodnota" .../>
<javac srcdir="..." destdir="..."/>
```

# Apache Ant – ukázka

```
<?xml version='1.0' encoding='utf-8'?>
<project name="IJA 05" default="compile"
         basedir=".">

  <description>
    IJA: Ukazka build.xml souboru
  </description>

  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>
  <property name="lib" location="lib"/>
```

# Apache Ant – ukázka

```
<target name="clean">
  <delete dir="${build}"/>
  <delete dir="${dist}"/>
</target>

<target name="compile" depends="clean"
        description="Compile">
  <mkdir dir="${build}"/>
  <javac srcdir="${src}"
         destdir="${build}"
         encoding="ISO8859-2"
         includes="ija/xml/XMLDemo.java"
       classpath="${lib}/dom4j-1.5.2.jar:${src}"/>
</target>
```

# Apache Ant – ukázka

```
<target name="jar" depends="clean,compile">
  <mkdir dir="${dist}"/>
    <jar destfile="${dist}/xmlDemo.jar"
         basedir="${build}">
      <manifest>
        <attribute name="Main-Class"
                   value="ija.xml.XMLDemo"/>
        <attribute name="Class-Path"
                   value="lib/dom4j-1.5.2.jar"/>
      </manifest>
    </jar>
  <mkdir dir="${dist}/lib"/>
  <copy file="${lib}/dom4j-1.5.2.jar"
        todir="${dist}/lib"/>
</target>
```

# Apache Ant – ukázka

```
  <target name="run">
    <java jar="${dist}/xmlDemo.jar"
          dir="${dist}"
          fork="true"/>
  </target>

</project>
```

# Apache Ant – použití

## Použití

- `ant [parametry] [target [target2 ... ]]`

## Parametry

- `propertyfile <soubor>` : definuje property ze zadaného souboru
- `-D<property>=<name>` : definice property
- `-buildfile <soubor>` : buildfile
- `-file <soubor>` : buildfile
- `-f <soubor>` : buildfile

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

# Apache Ant – úlohy (tasks)

## Vestavěné úlohy

- `javac`
- `java`
- `property`
- `javadoc`
- `delete`
- `move`
- `mkdir`
- `copy`
- `echo`
- `...`

`http://ant.apache.org/manual/index.html`

# Ladění programu

Pro ladění programů v Javě lze využít

- kontrolní tisky: `System.err.println(...)`
- řádkový debugger **`jdb`**
  - `http://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/jdb.html`
- 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

Použití

- `javac -cp junit.jar ...`
- `java -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 – `testNeco()`
  - 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);
    }
}
```

# Anotace

Annotations

- anotace definují dodatečné informace a data (metadata) k elementům programu (třída, metoda, ...)
- nemají přímý vliv na program
- zápis: `@jmeno_anotace`
- využití anotací
  - při kompilaci – detekce chyb, možnost generování dalšího kódu, ...
  - za běhu – nástroje a knihovny mohou na základě anotace přizpůsobit sémantiku
- lze využít předdefinované nebo definovat vlastní

# Anotace

Předdefinované anotace (`java.lang`)

- `@Deprecated`
  - náhrada za tag z dokumentačních komentářů
  - označuje, že daný element je zavrhovaný
- `@Override`
  - označení, že metoda v potomku předefinovává metodu z předka
  - pokud nic nepředefinovává, kompilátor odmítne třídu přeložit
- `@SuppressWarnings`
  - zamezí vypisování varování při překladu
  - parametr – třída varování (`unchecked`, `deprecation`)

# 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, ...)

# JUnit – Anotace

Používané anotace (`org.junit.*`)

- `@Test`
  - testovací metoda (Test Case)
- `@Before`
  - metoda, která se provede před každým testem
- `@After`
  - metoda, která se provede po každém testu
- `@Test(expected=IndexOutOfBoundsException.class)`
  - metoda by měla vracet výjimku `IndexOutOfBoundsException`

# JUnit – Assert

**`orj.junit.Assert`**

- `assertArrayEquals(...)`
- `assertEquals(...)`
- `assertFalse(...)`
- `assertTrue(...)`
- `assertNotNull(...)`
- `assertNull(...)`
- `assertNotSame(...)`
- `assertSame(...)`
- `assertThat(...)`
- `fail(...)`

# JUnit – Ukázka

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

public class TestHW {
    protected int x;

    @Before public void setUp() { x = 15; }

    @Test public void test01() {
        assertTrue(x>10);
    }
}
```

---

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