# Seminář Java

## Proudy, Zpracování XML

Radek Kočí

Fakulta informačních technologií VUT

Březen 2008

# Obsah

- Vstup/Výstup
  - koncepce proudů
  - soubory, adresáře
  - binární proudy (třídy **`InputStream`**, **`OutputStream`**)
  - znakové proudy (třídy **`Reader`**, **`Writer`**)
  - filtrované proudy
  - standardní vstup a výstup
- Komprimace
- Serializace
- Zpracování XML dokumentů

# Koncepce i/o operací

Balíčky

- `java.io` – základní knihovna pro práci s proudy
- `java.nio` – alternativní knihovna od verze 1.4+ (new IO)

I/O operace

- založeny na i/o proudech
- transparentní
- platformově nezávislé

Datový proud

- libovolný datový zdroj či cíl schopný zpřístupnit/přijmout libovolné datové bloky
- reprezentován objektem – ukrývání detailů (skutečné akce se skutečným i/o zařízením)

# Koncepce i/o operací

Proudy jako stavebnice

- každá třída reprezentuje jeden typ proudu (definuje jeho vlastnosti – operace)
- nové vlastnosti se dají získat vkládáním objektů (proudů) do sebe

```
...
is = new InputStream(...);
bis = new BufferedInputStream(is);
...
```

# Práce se soubory

Třída **`java.io.File`**

- ”obaluje” fyzické soubory (adresáře, linky) na disku
- speciality systémů souborů (transparentní odlišení)
  - **`char File.separatorChar`** – (′/′ nebo ′\′)
  - **`String File.separator`**
  - **`char File.pathSeparatorChar`** – (′ :′ nebo ′;′)
  - **`String File.pathSeparator`**
  - **`System.getProperty("user.dir")`** – adresář uživatele s jehož UID byl spuštěn proces JVM
- soubor (adresář) je reprezentován v objektu abstraktním **`pathname`**
- při přístupu k zařízení se konvertuje na systémově zavislé jméno
- vytvořením instance třídy **`File`** se k zařízení nepřistupuje!

# Třída `File`

Kostruktory

- **`File(String pathname)`** – vytvoří instanci **`File`** konvertováním řetězce **`pathname`** na abstraktní pathname
- **`File(String parent, String child)`** – vytvoří instanci **`File`** z řetězce **`parent`** a řetězce **`child`**
- **`File(File parent, String child)`** – vytvoří instanci **`File`** z abstraktního pathname objektu **`parent`** pathname a řetězce **`child`**
- **`File(URI uri)`** – vytvoří instanci **`File`** konvertováním daného **`file:URI`** na abstraktní pathname

# Třída `File`

Testovací operace

- `boolean exists()`
- `boolean isFile()`
- `boolean isDirectory()`
- `boolean canRead()`
- `boolean canWrite()`
- `...`

# Třída `File`

Vytvoření souboru (adresáře)

- **`boolean createNewFile()`** – vytvoří nový prázdný soubor podle abstraktního **`pathname`**; vrací true, pokud se oprace zdaří
- **`boolean mkdir()`** – vytvoří adresář . . .
- **`boolean mkdirs()`** – jako **`boolean mkdir()`**, vytváří i adresářovou strukturu, pokud je potřeba
- . . .

# Třída `File`

Vytvoření dočasného (temporary) souboru

- **`static File createTempFile(String prefix, String suffix)`** – vytvoří dočasný soubor ve standardním adresáři
- **`static File createTempFile(String prefix, String suffix, File directory)`** – vytvoří dočasný soubor v uvedeném adresáři

# Třída `File`

Další vlastnosti

- `boolean delete()`
- `boolean renameTo(File dest)`
- `long length()`
- `long lastModified()`
- `String getName()`
- `String getPath()`
- `String getAbsolutePath()`
- `String getParent()`
- ...

Všechny vlastnosti a operace

- viz API documentation

# Ukázka práce s adresáři – File

**`String[] list()`**

- vrací pole řetězců reprezentujících jména souborů a adresářů v adresáři daném abstraktním **`pathname`**

**`String[] list(FilenameFilter filter)`**

- vrací pole řetězců reprezentujících jména souborů a adresářů v adresáři daném abstraktním **`pathname`**, které odpovídají filtru

# Ukázka práce s adresáři – File

**`File[] listFiles()`**

- vrací pole abstraktních **`pathname`** reprezentujících soubory a adresáře v adresáři daném abstraktním **`pathname`**

**`File[] listFiles(FileFilter filter)`**

- vrací pole abstraktních **`pathname`** reprezentujících soubory a adresáře v adresáři daném abstraktním **`pathname`**, které odpovídají filtru

**`File[] listFiles(FilenameFilter filter)`**

- vrací pole abstraktních **`pathname`** reprezentujících soubory a adresáře v adresáři daném abstraktním **`pathname`**, které odpovídají filtru

# Ukázka práce s adresáři

Rozhraní **`FilenameFilter`**

- definuje jednu metodu
  **`boolean accept(File dir, String name)`**
  – test, zda uvedený soubor by měl být obsažen v seznamu

Rozhraní **`FileFilter`**

- definuje jednu metodu
  **`boolean accept(File pathname)`**
  – test, zda uvedený abstraktní **`pathname`** by měl být obsažen v seznamu

# Ukázka práce s adresáři

```
File cesta = new File(".");
String[] seznam;
seznam = cesta.list(new FilenameFilter() {
    public boolean accept(File adr, String cesta)
      {
        String soubor = new File(cesta).getName();
        return soubor.indexOf("x") != -1;
      }
    });
}

for(String name : seznam) {
  System.out.println(name);
}
```

# Binární proudy

Vstupní (odvozeny od abstraktní třídy **`InputStream`**)

- pole bytů
- soubor
- **`String`**
- roura
- ...

Výstupní (odvozeny od abstraktní třídy **`OutputStream`**)

- pole bytů
- soubor
- ...

# Vstupní binární proudy

Operace

- `void close()` – uzavře proud, uvolní zdroje
- `abstract int read()` – čte byte, pokud nelze, vrací -1
- `int read(byte[] b)` – čte `b.length` bytů
- `int read(byte[] b, int off, int len)` – čte `len` bytů z proudu od pozice `off`
- `long skip(long n)` – přeskočí `n` bytů
- `void mark(int readlimit)` – poznačí aktuální pozici (`readlimit` je počet bytů, které lze přečíst, než se značka stane nevalidní)
- `void reset()` – obnovení pozice uchované při volání `mark()`
- `boolean markSupported()`

# Vstupní binární proudy

Odvozené od **`FilterInputStream`** $\longrightarrow$ **`InputStream`**

- **`java.io.BufferedInputStream`** – proud s vyrovnávací pamětí
- **`java.util.zip.CheckedInputStream`** – proud s kontrolním součtem
- **`javax.crypto.CipherInputStream`** – proud dešifrující data
- **`java.io.DataInputStream`** – proud s metodami pro čtení hodnot primitivních datových typů
- . . .

# Výstupní binární proudy

- analogicky k vstupním
- **`InputStream`** $\Rightarrow$ **`OutputStream`**
- ...

# Ukázka

```
public static void main(String[] args) {
  try {
    FileOutputStream fos =
                new FileOutputStream("t.tmp");
    ObjectOutputStream oos =
                  new ObjectOutputStream(fos);
    oos.writeInt(12345);
    oos.writeObject("Today");
    oos.writeObject(new Date());
    oos.close();

    ...
```

# Ukázka

```
      FileInputStream fis =
                    new FileInputStream("t.tmp");
      ObjectInputStream ois =
                     new ObjectInputStream(fis);
      int i = ois.readInt();
      String today = (String) ois.readObject();
      Date date = (Date) ois.readObject();
      ois.close();

      System.out.println(i + today + date);

    } catch (Exception ex) {
      ex.printStackTrace();
    }
}
```

# Znakové proudy

Vstupní (odvozeny od abstraktní třídy **`Reader`**)

- **`BufferedReader`**
- **`PipedReader`**
- **`InputStreamReader`**
- **`FileReader`** $\longrightarrow$ **`InputStreamReader`**
- ...

Výstupní (odvozeny od abstraktní třídy **`Writer`**)

- **`BufferedReader`**
- **`PipedWriter`**
- **`InputStreamWriter`**
- **`FileWriter`** $\longrightarrow$ **`InputStreamWriter`**
- ...

# Konverze mezi proudy

Z binárního proudu (vstupní/výstupní) lze vytvořit proud znakový (vstupní/výstupní)

```
// vytvoříme binární vstupní proud
InputStream is = ...

// z is vytvoříme znakový proud (pro
// dekódování se použije std.  znaková sada)
Reader isr = new InputStreamReader(is);

// znakové sady jsou v balíku java.nio
Charset chrs =
        java.nio.Charset.forName("ISO-8859-2");

// z is vytvoříme znakový proud (pro
// dekódování se použije jiná znaková sada)
Reader isr2 = new InputStreamReader(is, chrs);
```

# Standardní vstup a výstup

Ve třídě **`java.lang.System`**

- **`static InputStream in`**
- **`static PrintStream out`**
- **`static PrintStream err`**

Změna standardních proudů

- **`setIn(InputStream in)`**
- **`setOut(PrintStream out)`**
- **`setErr(PrintStream err)`**

# Příklad – čtení ze std. vstupu

```
public class ReadIn {
  public static void main(String[] args) {
    try {
      BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in));

      String line;
      while((line = in.readLine()).length() != 0) {
        System.out.println(line);
      }
    }
    catch(IOException ioe) {
      ioe.printStackTrace(System.err);
    }
  }
}
```

## Package **java.util.zip**

```
BufferedReader in = new BufferedReader(
      new FileReader(args[0]));

BufferedOutputStream out =
  new BufferedOutputStream(new GZIPOutputStream(
    new FileOutputStream(args[1])));

int b;
while((b = in.read()) != -1)
   out.write(b);

in.close();
out.close();
```

# Serializace objektů

Serializace

- perzistence objektů
- získání reprezentace objektu jako sekvence bytů
- uložení reprezentace do souboru

Deserializace

- rekonstrukce serializovaného objektu

# Serializace objektů

Serializovaný objekt

- musí implementovat rozhraní **`java.io.Serializable`**
  - nemá žádné metody, slouží pouze pro identifikaci serializovatelného objektu
  - všechny podřídy třídy, která implementuje toto rozhraní, jsou také serializovatelné
- proměnné objektu, které nemají být serializovány, musí být označeny modifikátorem **`transient`**
- pokud je při (de)serializace nutné speciální chování, musí objekt definovat tyto metody

```
private void readObject(ObjectInputStream is)
private void writeObject(ObjectOutputStream os)
```

- viz API pro **`java.io.Serializable`**

# Serializace objektů – příklad

```
class Clovek implements Serializable {
    String jmeno;
    protected Clovek() { }
    public Clovek(String jmeno) {
        this.jmeno = jmeno;
    }
    public String info() {
        return jmeno;
    }
}
```

# Serializace objektů – příklad

```
private static final String soubor = "clovek.dat";

public static void serializuj(Clovek p)
throws Exception
{
    FileOutputStream fos =
            new FileOutputStream(soubor);
    ObjectOutputStream oos =
            new ObjectOutputStream(fos);
    oos.writeObject(p);
    fos.close();
}
```

# Serializace objektů – příklad

```
public static Clovek deserializuj()
throws Exception
{
    FileInputStream fis =
            new FileInputStream(soubor);
    ObjectInputStream ois =
            new ObjectInputStream(fis);
    Clovek p = (Clovek)ois.readObject();
    fis.close();
    return p;
}
```

# Práce s XML

XML

- je standard konsorcia W3C jak vytvářet značkovací jazyky
- ideově vychází ze zhruba o deset let mladšího standardu SGML (Structure Generalized Markup Language).
- XML není jeden konkrétní značkovací jazyk
  - je to specifikace určující, jak mají vypadat značkovací jazyky
  - je to metajazyk
  - konceptuálně jde o zjednodušení SGML standardu, aby se usnadnila práce tvůrcům parserů (analyzátorů) a aplikací
- se základním standardem úzce souvisejí další, např. XML Namespaces, XInclude, XML Base.
- společně s dalšími standardy (XSLT, XSL-FO, XHTML, CSS...) tvoří ”rodinu” standardů XML.

# Struktura XML dokumentu

Logická

- *dokument* se člení na elementy (jedna z nich je kořenová), atributy elementů, textové uzly v elementech, intrukce pro zpracování, notace, komentáře
- *dokument* je reprezentován stromem elementů

Fyzická

- jeden logický dokument může být uložen ve více fyzických jednotkách (entitách); vždy alespoň v jedné – document entity.
- **<!ENTITY kap1 SYSTEM "kapitola1.xml">**
- **<!ENTITY swn "Softwarové noviny">**

# Zpracování XML dokumentu

Hlavní typy rozhraní pro zpracování XML dat:

- Stromově orientovaná rozhraní (Tree-based API)
- Rozhraní založená na událostech (Event-based API)
- Rozhraní založená na ”vytahování” událostí/prvků z dokumentu (Pull API)

Implementace pro Javu

- parsery
- součást JAXP (Java API for XML Processing) `http://java.sun.com/xml/jaxp/index.html`
- nezávislé – např. `http://dom4j.org`

# Tree-based API

Mapují XML dokument na stromovou strukturu v paměti

- dovolují libovolně procházet vzniklý strom;
- nejznámější je Document Object Model (DOM) konsorcia W3C, `viz http://www.w3.org/DOM`

Modely specifické pro konkrétní prostředí

- pro Javu: JDOM – `http://jdom.org`
- pro Javu: dom4j – `http://dom4j.org`
- pro Python: 4Suite – `http://4suite.org`

# Event-based API

Při analýze dokumentu generují události (podle toho, kde se nacházejí)

- zpracovávající aplikace implementuje obsluhu těchto událostí (pomocí metod – callback)
- událostmi řízená rozhraní jsou ”nižší úrovně” než stromová (pro aplikaci zůstává ”více práce”)
- jsou úspornější na paměť (i čas), samotná analýza nevytváří trvalé objekty

# Event-based API

Událost je . . .

- začátek a konec dokumentu (start document, end document)
- komentář (comment)
- odkaz na entitu (entity reference)
- . . .

Nejznámějším rozhraním je SAX (`http://www.saxproject.org`)

# Event-based API

Ukazka

```
<?xml version="1.0"?>
<doc>
   <para>Hello, world!</para>
</doc>
```

---

Sled udalosti:

```
start document
start element: doc {seznam atributu: prazdny}
start element: para {seznam atributu: prazdny}
characters: Hello, world!
end element: para
end element: doc
end document
```

# Zdroje informací

- World Wide Web Consortium (W3C)
  `http://www.w3.org/`
- XML Startkabel (EN/NL): aktuality, odkazy
  `http://xml.startkabel.nl`
- Soubor tutoriálů a on-line referencí v mnoha jazycích
  `http://zvon.org`
- O'Reilly XML.COM: články, tutoriály atd. na vysoké technické úrovni
  `http://xml.com`
- Free XML Software (L. M. Garshol): kolekce odkazů na nekomerční XML software
  `http://www.garshol.priv.no/download/xmltools/`
- Kolekce odkazů na obecný XML software (i komerční)
  `http://xmlsoftware.com`
- J. Kosek: `http://kosek.cz/xml/index.html`