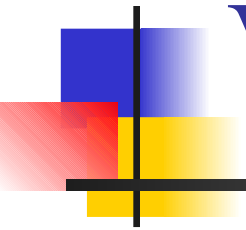
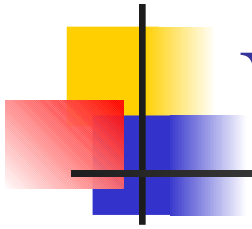


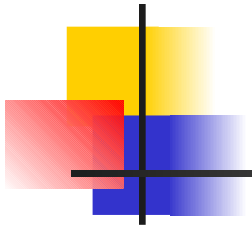
Vstup/Výstup



Vstup/Výstup

- Je nutné rozlišovat práci se souborovým systémem před Java SE 7 a po Java SE 7.
  - Do Java SE 7 se používal balíček `java.io`
  - Od Java SE 7 navíc existuje balíček `java.nio`, který je vhodné v maximální míře využívat (a dá se kombinovat s balíčkem `java.io`)

- Navíc existuje knihovna Apache Commons IO, ve které je další spousta užitečných metod pro práci se soubory:
  - http://commons.apache.org/proper/commons-io/



Práce se souborovým systémem od Java SE 7

Práce s textovým souborem

- Zápis dat do textového souboru:

```
List<String> content = new ArrayList<>();
content.add("Hello");
content.add("+ěščřžýáié");
Files.write(Paths.get("c:/bar.txt"), content,
            StandardCharsets.UTF_8, StandardOpenOption.CREATE);
```

Poznámka:
Charset pro kódování Windows-1250:
`Charset.forName("windows-1250")`

- Čtení dat z textového souboru:

```
List<String> lines = Files.readAllLines(Paths.get("c:/baz.txt"),
                                        StandardCharsets.UTF_8);
for (String line : lines) {
   System.out.println(line);
}
```

Práce se soubory / adresáři na disku

- Kopírování souboru:

```
Files.copy(Paths.get("c:/bar.txt"), Paths.get("c:/baz.txt"),
           StandardCopyOption.REPLACE_EXISTING);
```

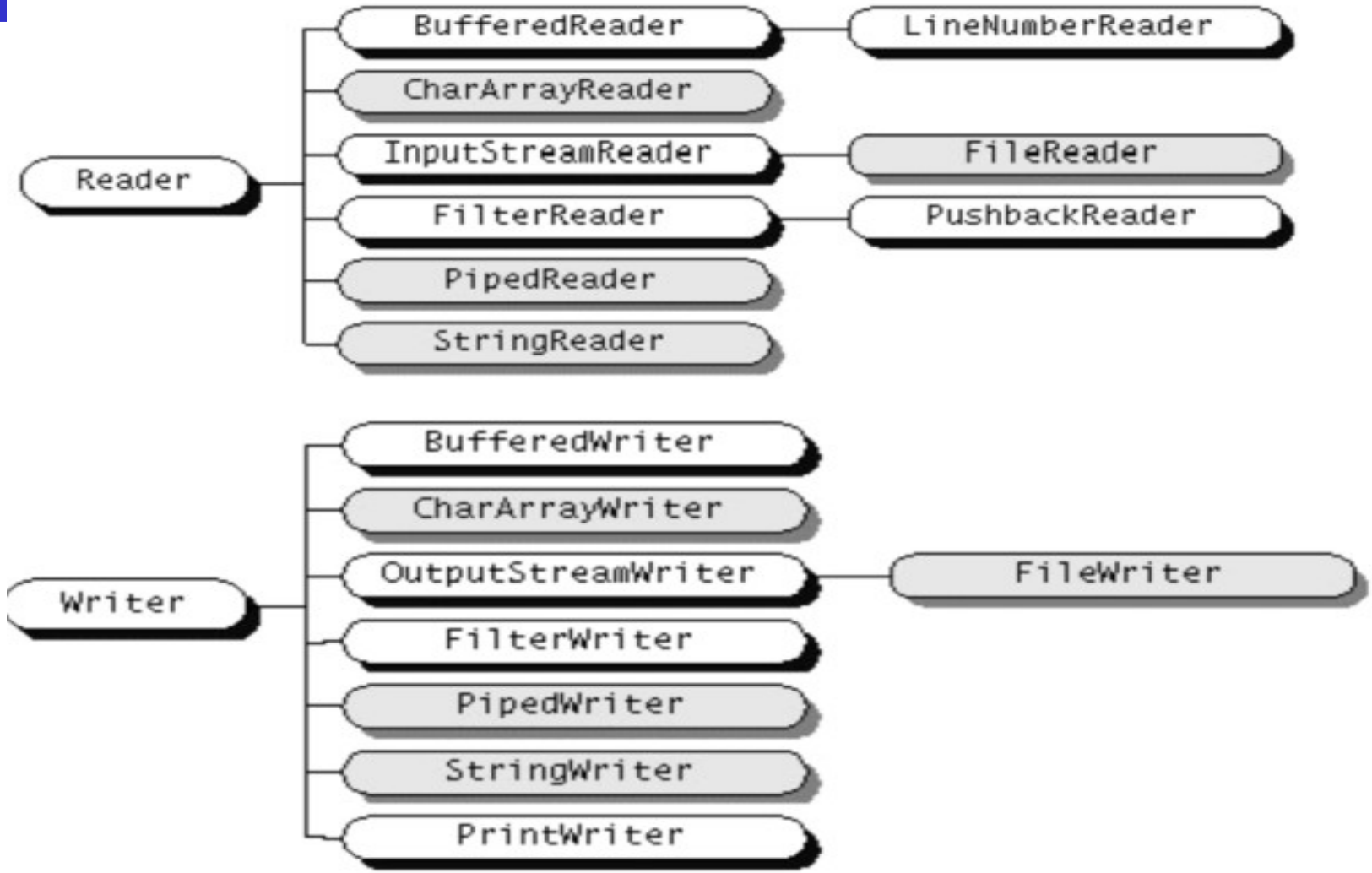
- Mazání souboru:

```
Files.delete(Paths.get("c:/bar.txt"));
Files.deleteIfExists(Paths.get("c:/baz.txt"));
```

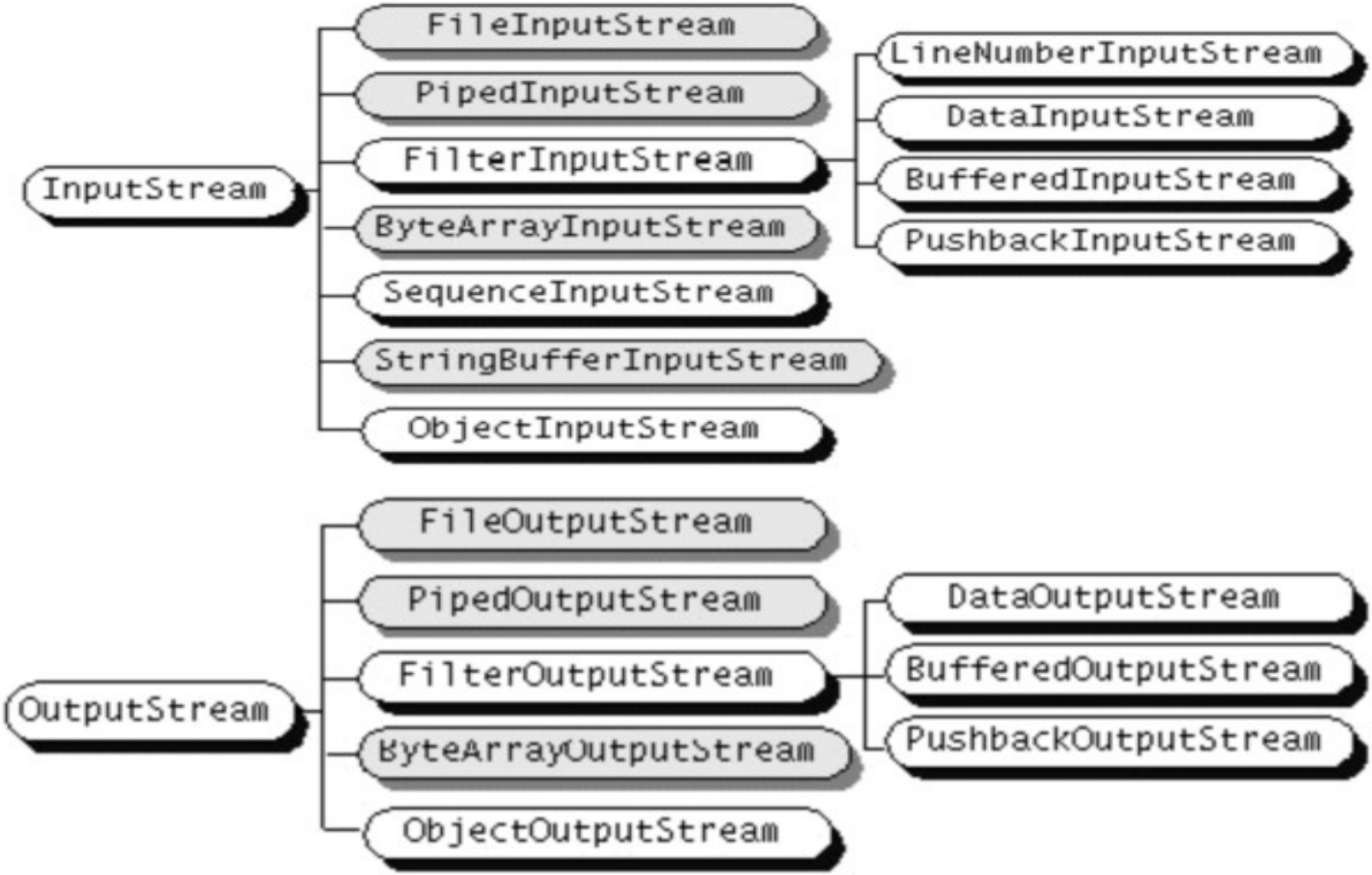
- V třídě `java.nio.file.Files` je celá řada užitečných metod pro práci se soubory a adresářovou strukturou disku:
  - http://docs.oracle.com/javase/7/docs/api/java/nio/file/Files.html

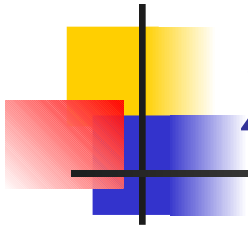
Práce se souborovým systémem před Java SE 7

Znakové proudy



Bytové proudy





Znakové vs. Bytové proudy

- Znakové proudy je vhodné použít pro čtení/zápis znaků
- Bytové proudy jsou vhodné pro čtení/zápis binárních dat (zvuk, obraz, objekty, …)
- Třídy `Reader, Writer, InputStream, OutputStream` se nepoužívají. Používají se jejich potomci.
- Velké množství různých druhů přístupů k souboru. Na první pohled je to komplikované, ale tento přístup má výhodu flexibility (stejný přístup je k datovému souboru, ale i proudu dat přenášenému přes Internet, je možné přímo číst data ze ZIP archivu atd.)

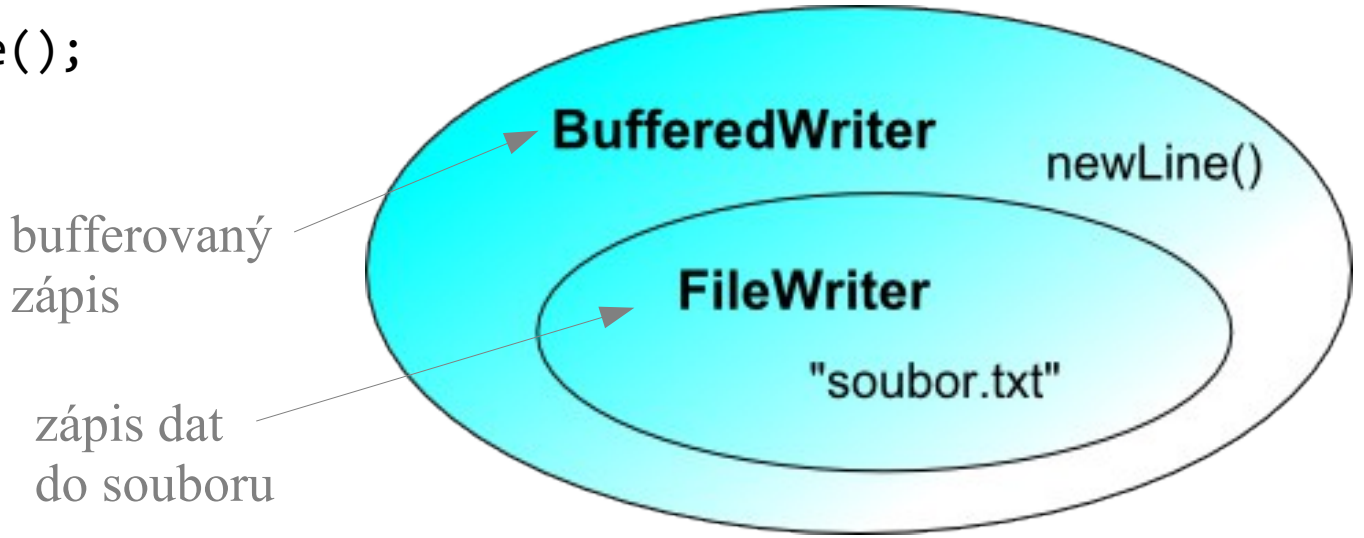
Nejjednodušší čtení a zápis textového souboru I.

- Zápis dat do textového souboru:

```
BufferedWriter w = new BufferedWriter(
                       new FileWriter("soubor.txt"));
w.write("řádek 1");
w.newLine();
w.write("další řádek");
w.close();
```

Pokud soubor neexistuje, tak se vytvoří. Pokud existuje, tak se přepíše. FileWriter má ještě další konstruktor, v němž je možné to změnit.

V jakém kódování se data do souboru uloží? To závisí na platformě, na Windows to je windows-1250, jinde většinou UTF-8

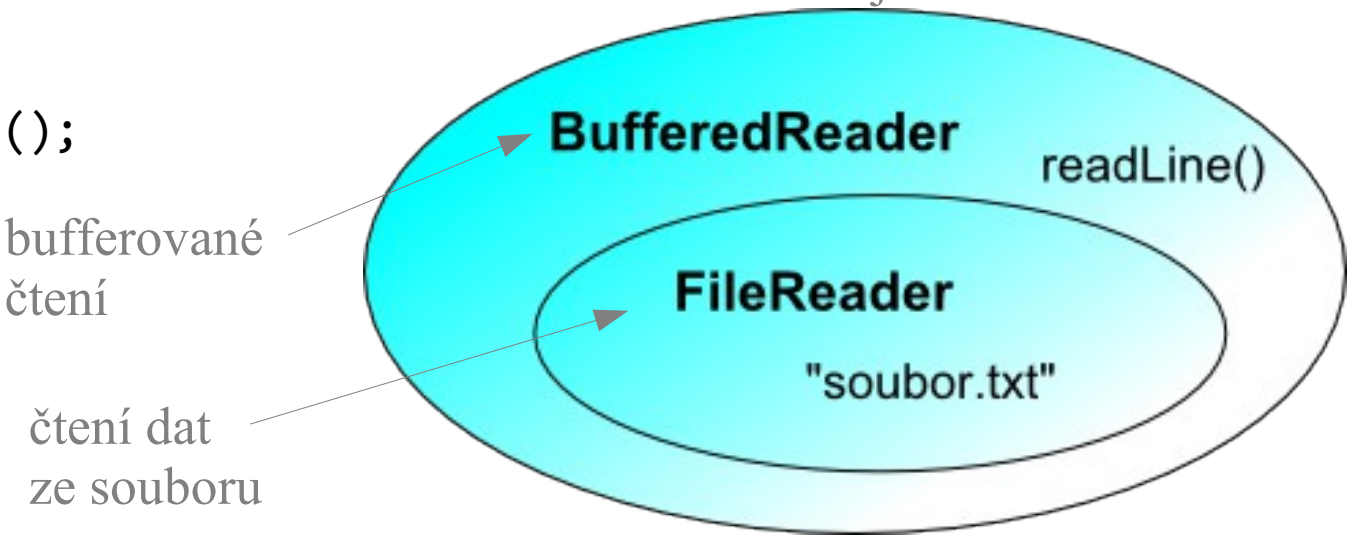


Nejjednodušší čtení a zápis textového souboru II.

- Čtení dat z textového souboru:

```
BufferedReader r = new BufferedReader(
                        new FileReader("soubor.txt"));
String radka = null;
while((radka = r.readLine()) != null) {
   System.out.println(radka);
}
r.close();
```

Soubor musí mít kódování windows-1250 na OS Windows, jinde většinou UTF-8



Čtení a zápis textového souboru s definovaným kódováním

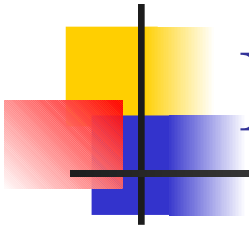
- Pro zápis dat do souboru s kódováním UTF-8:

```
BufferedWriter w = new BufferedWriter(
                       new OutputStreamWriter(
                         new FileOutputStream("soubor.txt"),
                           "utf-8"));
```

- Pro čtení dat ze souboru s kódováním UTF-8:

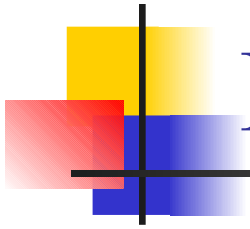
```
BufferedReader r = new BufferedReader(
                       new InputStreamReader(
                         new FileInputStream("soubor.txt"),
                           "utf-8"));
```

- Nejčastěji používaná kódování v našem regionu:
  - `utf-8, windows-1250, iso-8859-2`



Metody proudů

- Voláním konstruktoru se otevírá vstup/výstup.
- Uzavření proudu dat se provádí pomocí metody `close()`. Nezapomeňte ji zavolat, jinak se například při zápisu dat do souboru některé znaky nemusí do souboru zapsat!
  - Při bufferovaném zápisu dat do souboru se pro vyšší výkon používá buffer, do kterého se při zavolání metody `write()` ukládají data a jenom při zaplnění bufferu nebo při zavolání metody `close()` se provede fyzický zápis dat do souboru.
- `Reader, InputStream` obsahují metodu `read()` pro čtení znaků/bytů ze streamu.
- `Writer, OutputStream` obsahují metodu `write()` pro zápis znaků/bytů do streamu.



Práce se souborovým systémem

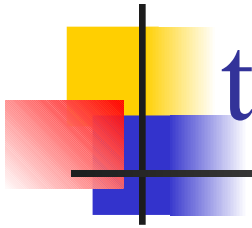
- Do Java SE 7 se pro základní práci se souborovým systémem používala třída `File` (v balíčku `java.io`):

```
// vytvoří nový adresář
File adresar = new File("conf/");
adresar.mkdir();

// smaže soubor
File soubor = new File("soubor.txt");
soubor.delete();
```

- Pro pokročilejší práci (kopírování souborů, rekurzivní mazání adresářů apod.) se pak používala knihovna Apache Commons IO.

Správná konstrukce bloku try-catch-finally (před Java SE 7)



```
FileReader reader = null;
try {
   reader = new FileReader("soubor.txt");
   // práce se souborem
} catch (FileNotFoundException e) {
   e.printStackTrace();
} finally {
   try {
      if (reader != null) {
         reader.close();
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Správná konstrukce bloku try-with-resources (od Java SE 7)

```
try (BufferedReader reader = new BufferedReader(new FileReader("soubor.txt"))) {
   // práce se souborem
} catch (IOException e) {
   e.printStackTrace();
}
```

- Nebo ještě lépe:

```
try (BufferedReader reader = Files.newBufferedReader(Paths.get("soubor.txt"))) {
   // práce se souborem
} catch (IOException e) {
   e.printStackTrace();
}
```

- Java se postará o uzavření proudu (zavolání metody `close()`)

Zachycení více typů výjimek

- Do Java SE 7 bylo nutné pro zachycení výjimek typu `IOException` a `SQLException` použít zápis:

```
catch (IOException ex) {
     logger.log(ex);
catch (SQLException ex) {
     logger.log(ex);
}
```

- Od Java SE 7 je možné v catch bloku zachytit více typů výjimek:

```
catch (IOException|SQLException ex) {
    logger.log(ex);
}
```

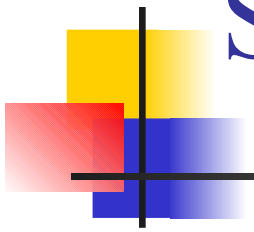
Formátovaný výstup

- Formátování čísel / data / času se provádí pomocí třídy `Formatter`. Formát textu je inspirován `printf` příkazem z jazyka C:

```
Formatter formatter = new Formatter();
// Výpis čísla pí na šířku 12 znaků
// (včetně desetinné čárky),
// 10 desetinných míst
formatter.format("PI = %12.10f", Math.PI);
System.out.println(formatter);
```

- Více informací naleznete v dokumentaci.

Specializované typy souborů I.

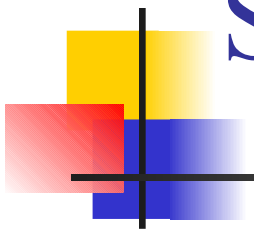


- Existují známé typy souborů, pro které byly vytvořeny specializované knihovny:
  - Properties soubory
    - Jedná se o soubory, ve kterých jsou data ve formátu:

      ```
      klic=hodnota
      ```

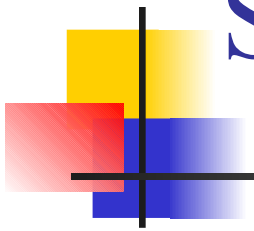
    - Používají se pro uchování konfiguračních údajů.
    - http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html
  - CSV (Comma Separated Value) soubory
    - Soubory, ve kterých jsou jednotlivé hodnoty oddělené oddělovačem (například čárka).
    - http://commons.apache.org/proper/commons-csv/

Specializované typy souborů II.



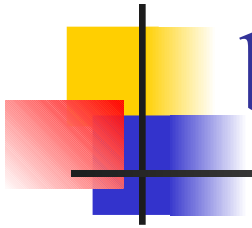
- XML soubory
  - Knihovna JAXB (od Java SE 6 součástí Java SE), SAX, DOM a řada dalších knihoven.
- JSON soubory
  - JSON je velice populární formát při práci s JavaScriptem nebo v různých NoSQL databázích (MongoDB, Neo4J apod.).
  - https://github.com/FasterXML/jackson
- MS Word / Excel
  - http://poi.apache.org/

Specializované typy souborů III.



- Generování PDF dokumentů
  - K tomu existuje řada knihoven, je nutné dát pozor na licencování, ne všechny jsou zdarma i pro komerční použití.
  - Nejpopulárnější je knihovna iText, která byla do verze 4.2.1 zdarma i pro komerční použití:
    - http://itextpdf.com/
    - http://mvnrepository.com/artifact/com.lowagie/itext/4.2.1
- SVG (Scalable Vector Graphics)
  - http://xmlgraphics.apache.org/
- … cokoli Vás napadne, pro to nejspíš existuje knihovna v Javě.

Přístup k vlastním datům v Jar balíčku

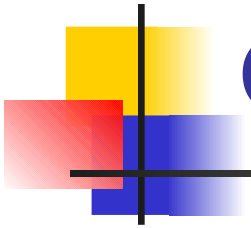


getClass().getResource(„název“);

- Vrátí se URL daného souboru
- „název“ je ve formátu: /slozka/podslozka/soubor
- „název“ vždy začíná lomítkem!
- Správnou cestu k souboru si lze ověřit/zjistit rozbalením Jar archivu

getClass().getResourceAsStream(„název“);

- Vrátí se InputStream daného souboru
- Častěji používané kvůli tomu, že se přímo vrátí stream, který je možné ihned používat



Omezení resource v jar souboru

- Soubor není možné měnit
- Je možné ze souboru číst pouze pomocí InputStream! => Není možné použít přístup pro čtení z textového souboru!
- Soubor je a není skrytý pro uživatele
  - Není standardně vidět
  - Ale pokud uživatel ví, že jar soubor je zip archiv, poté může přečíst obsah tohoto souboru.
  - Všechny třídy v jar souboru jsou však ve zkompilované (binární) podobě.