

14. Vstupy a výstupy v Javě

- Koncepte I/O proudů v Javě, skládání (obalování vlastnostmi)
- Práce se soubory a adresáři, třída `File`
- Binární proudy, třídy `InputStream`, `OutputStream`
- Znakové proudy, třídy `Reader`, `Writer`
- Serializace objektů

Koncepte vstupně/výstupních operací v Javě

založeny na v/v proudech

plně **platformově nezávislé**

V/V proudy jsou

- **znakové** (`Reader/Writer`) nebo
- **binární** (`Stream`)

koncipovány jako "stavebnice" - lze vkládat do sebe a tím přidávat vlastnosti, např.

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

Téměř vše ze vstupních/výstupních tříd a rozhraní je v balíku `java.io`.

počínaje J2SDK1.4 se rozvíjí alternativní balík - `java.nio` (**New I/O**)

Blíže viz dokumentace API balíků [java.io](#), [java.nio](#).

Práce se soubory

vše je opět v balíku `java.io`

základem je třída `java.io.File` - nositel jména souboru, jakási "brána" k fyzickým souborům na disku.

používá se jak pro soubory, tak adresáře, linky i soubory identifikované UNC jmény (\\počítač\adresář...)

opět plně platformově nezávislé

na odstínění odlišností jednotlivých systémů souborů lze použít vlastností (uvádíme jejich hodnoty pro JVM pod systémem MS Windows):

- `File.separatorChar` \ - jako `char`
- `File.separator` \ - jako `String`
- `File.pathSeparatorChar` ; - jako `char`
- `File.pathSeparator` ; - jako `String`
- `System.getProperty("user.dir")` - adresář uživatele, pod jehož UID je proces JVM spuštěn

Třída `File`

Vytvoření konstruktorem - máme několik možností:

```
new File(String filename)
```

vytvoří v aktuálním adresáři soubor s názvem **filename**
new File(File **baseDir**, String **filename**)
vytvoří v adresáři **baseDir** soubor s názvem **filename**
new File(String **baseDirName**, String **filename**)
vytvoří v adresáři **se jménem baseDirName** soubor s názvem **filename**
new File(URL **url**)
vytvoří soubor se (souborovým - file:) URL **url**
Testy existence a povahy souboru:
boolean **exists()**
test na existenci souboru (nebo adresáře)
boolean **isFile()**
test, zda jde o soubor (tj. ne adresář)
boolean **isDirectory()**
test, zda jde o adresář
Test práv ke čtení/zápisu:
boolean **canRead()**
test, zda lze soubor číst
boolean **canWrite()**
test, zda lze do souboru zapisovat

Třída File (2)

Vytvoření souboru nebo adresáře:

boolean **createNewFile()**
(pro soubor) vrací `true`, když se podaří **soubor** vytvořit
boolean **mkdir()**
(pro adresář) vrací `true`, když se podaří adresář vytvořit
boolean **mkdirs()**
navíc si dotvoří i příp. neexistující adresáře na cestě
Vytvoření dočasného (temporary) souboru:
static File **createTempFile**(String **prefix**, String **suffix**)
vytvoří dočasný soubor ve standardním pro to určeném adresáři
(např. `c:\temp`) s uvedeným prefixem a sufixem názvu
static File **createTempFile**(String **prefix**, String **suffix**, File **directory**)
dtto, ale vytvoří dočasný soubor v adr. **directory**

Zrušení:

boolean **delete()**
zrušení souboru nebo adresáře
Přejmenování (ne přesun mezi adresáři!):
boolean **renameTo**(File **dest**)
přejmenuje soubor nebo adresář

Třída File (3)

Další vlastnosti:

long **length()**
délka (velikost) souboru v bajtech

long **lastModified()**

čas poslední modifikace v ms od začátku éry - podobně jako systémový čas vrácený `System.currentTimeMillis()`.

String **getName()**

jen jméno souboru (tj. poslední část cesty)

String **getPath()**

celá cesta k souboru i se jménem

String **getAbsolutePath()**

absolutní cesta k souboru i se jménem

String **getParent()**

adresář, v němž je soubor nebo adresář obsažen

Blíže viz [dokumentace API třídy File](#).

Práce s adresáři

Klíčem je opět třída `File` - použitelná i pro adresáře

Jak např. získat (filtrovaný) seznam souborů v adresáři?

pomocí metody `File[] listFiles(FileFilter ff)` nebo podobně

`File[] listFiles(FileNameFilter fnf):FileFilter je rozhraní s jedinou metodou boolean accept(File pathname), obdobně FileNameFilter, viz Popis API java.io.FileNameFilter`

Práce s binárními proudy

Vstupní jsou odvozeny od abstraktní třídy `InputStream`

Výstupní jsou odvozeny od abstraktní třídy `OutputStream`

Vstupní binární proudy

Uvedené metody, kromě `abstract byte read()`, nemusejí být nutně v neabstraktní podtřídě překryty.

void **close()**

uzavře proud a uvolní příslušné zdroje (systémové "file handles" apod.)

void **mark(int readlimit)**

poznačí si aktuální pozici (později se lze vrátit zpět pomocí `reset()`)...

boolean **markSupported()**

...ale jen když platí tohle

abstract int **read()**

přečte bajt (0-255 pokud OK; jinak -1, když už není možné přečíst)

int **read(byte[] b)**

přečte pole bajtů

int **read(byte[] b, int off, int len)**

přečte pole bajtů se specifikací délky a pozice plnění pole b

void **reset()**

vrátí se ke značce nastavené metodou `mark(int)`

long **skip**(long n)
přeskočí zadaný počte bajtů

Důležité neabstraktní třídy odvozené od `InputStream`

`java.io.FilterInputStream` - je bázová třída k odvozování všech vstupních proudů přidávajících vlastnost/schopnost filtrovat poskytnutý vstupní proud.

Příklady filtrů (ne všechny jsou v `java.io`):

`BufferedInputStream`

proud s vyrovnávací pamětí (je možno specifikovat její optimální velikost)

`java.util.zip.CheckedInputStream`

proud s kontrolním součtem (např. CRC32)

`javax.crypto.CipherInputStream`

proud dešifrující data ze vstupu

`DataInputStream`

má metody pro čtení hodnot primitivních typů, např. `float readFloat()`

`java.security.DigestInputStream`

počítá současně i haš (digest) čtených dat, použitý algoritmus lze nastavit

`java.util.zip.InflaterInputStream`

dekomprimuje (např. GZIPem) zabalený vstupní proud (má ještě specializované podtřídy)

`LineNumberInputStream`

doplňuje informaci o tom, ze kterého řádku vstupu čteme (zavrhaná - **deprecated** - třída)

`ProgressMonitorInputStream`

přidává schopnost informovat o průběhu čtení z proudu

`PushbackInputStream`

do proudu lze data vracet zpět

Další vstupní proudy

Příklad rekonstrukce objektů ze souborů

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

`javax.sound.sampled.AudioInputStream`

vstupní proud zvukových dat

`ByteArrayInputStream`

proud dat čtených z pole bajtů

`PipedInputStream`

roura napojená na "protilehlý" `PipedOutputStream`

`SequenceInputStream`

proud vzniklý spojením více podřízených proudů do jednoho

virtuálního

ObjectInputStream

proud na čtení serializovaných objektů

Práce se znakovými proudy

základem je abstraktní třída `Reader`, konkrétními implementacemi jsou:

- `BufferedReader`, `CharArrayReader`, `InputStreamReader`,
`PipedReader`, `StringReader`
- `LineNumberReader`, `FileReader`, `PushbackReader`

Výstupní proudy

nebudeme důkladně probírat všechny typy

principy:

- jedná se o protějšky k vstupním proudům, názvy jsou konstruovány analogicky (např. `FileReader` -> `FileWriter`)
- místo generických metod `read` mají `write(...)`

Příklady:

`PrintStream`

poskytuje metody pro pohodlný zápis hodnot primitivních typů a řetězců - příkladem jsou `System.out` a `System.err`

`PrintWriter`

poskytuje metody pro pohodlný zápis hodnot primitivních typů a řetězců

Konverze: znakové <-> binární proudy

Ze vstupního binárního proudu `InputStream` (čili každého) je možné vytvořit znakový `Reader` pomocí

```
// nejprve binární vstupní proud - toho kódování znaků nezajímá
InputStream is = ...

// znakový proud isr
// použije pro dekódování standardní znakovou sadu
Reader isr = new InputStreamReader(is);

// sady jsou definovány v balíku
java.nio

Charset chrs = java.nio.Charset.forName("ISO-8859-2");

// znakový proud isr2
// použije pro dekódování jinou znakovou sadu
Reader isr2 = new InputStreamReader(is, chrs);
```

Podporované názvy znakových sad naleznete na webu [IANA Charsets](#).

Obdobně pro výstupní proudy - lze vytvořit `Writer` z `OutputStream`.

Serializace objektů

- nebudeme podrobně studovat, zatím stačí vědět, že:
 - **serializace objektů** je postup, jak z objektu vytvořit sekvenci bajtů persistentně uložitelnou na paměťové médium (disk) a později restaurovatelnou do podoby výchozího javového objektu.
 - **deserializace** je právě zpětná rekonstrukce objektu
- aby objekt bylo možno serializovat, musí implementovat (prázdné) rozhraní `java.io.Serializable`
- proměnné objektu, které nemají být serializovány, musí být označeny modifikátorem - klíčovým slovem - **transient**
- pokud požaduje "speciální chování" při de/serializaci, musí objekt definovat metody
 - `private void readObject (java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException`
 - `private void writeObject (java.io.ObjectOutputStream stream) throws IOException`
- metody:
 - `DataOutputStream.writeObject (Object o)`

Odkazy

Tutoriály k Java I/O: [kapitola z Sun Java Tutorial](#)

Demo programy na serializaci (z učebnice): [Serializace objektů](#)