

Soubory

Jan Faigl

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

Přednáška 12

A0B36PR1 – Programování 1

Část 1 – Soubory

Soubory

Práce se soubory

Čtení a zápis souboru v Javě

Binární soubory

Textové soubory

Část I

Soubory

Soubory a organizace dat v souborovém systému

- Soubor je množina údajů uložená ve vnější paměti počítače

Obvykle na pevném disku

- Typické operace pro soubor jsou:

1. Otevření souboru
2. Čtení dat
3. Zápis dat
4. Zavření souboru

- Přístup k datům (údajům) v souboru může být

- Sekvenční (postupný)

Postupné čtení nebo zápis dat do souboru

- Náhodný (adresovatelný)

Umožňuje adresovat libovolné místo v souboru podobně jako při přístupu do pole

- Způsob přístup k údajům v souboru není zakódován v souboru, ale je dán programem

Podobně také případ, zdali soubor „chápeme“ jako textový nebo binární.

Adresa (cesta) k souboru

- Soubory jsou uloženy v souborovém systému
- Soubory organizujeme do složek (adresářů), které tvoří hierarchii adresářů a souborů tvořící stromovou strukturu

Lze sice vytvořit i cykly, zpravidla je to však speciální případ.

- Souborový systém představuje „adresovatelný“ prostor, kde
- ke každému souboru existuje „adresa“ identifikující v jakém adresáři (složce) se soubor nachází
- Adresa je složena ze jmen jednotlivých adresářů oddělených znakem /

Podobně jako URL

např. `/usr/local/bin/netbeans-8.0` – představuje cestu k

- `netbeans-8.0` – soubor pro spuštění programu Netbeans
- `bin` – adresář v adresáři `local`
- `local` – adresáři v adresáři `usr`
- `/` – kořenový adresář

Umístění souboru tak můžeme jednoznačně určit

Umístění souboru – absolutní a relativní cesta

- Adresa absolutního umístění souboru v systému souborů začíná kořenovým adresářem /
- Cesta k souboru může být také relativní vzhledem k nějakému pracovnímu (např. projektovému) adresáři
- Speciální význam mají adresáře
 - .. – odkazuje do adresáře o úroveň výše
 - . – je aktuální adresář
- Příklady
 - `/usr/local/bin/netbeans`
 - Relativní cesta vzhledem k `/usr/local/tmp` je `../bin/netbeans`
 - Relativní cesta vzhledem k `/usr/local/bin` je
 - `netbeans`
 - `./netbeans`

Typy souborů

Podle způsobu kódování informace v souboru rozlišujeme:

■ Textové soubory

- Přímě čitelné a jednoduše editovatelné

Běžným textovým editorem

■ Binární soubory

- Zpravidla potřebujeme specializovaný program pro čtení, zápis a modifikaci souboru
- Přístup k souboru tak spíše realizujeme prostřednictvím programového rozhraní

V obou případech je pro výměnu souboru a jejich použitelnost v jiných programech klíčový konkrétní způsob organizace údajů a informací uložených v souborech

Používání standardních formátů a to jak textových (např. XML, HTML, JSON, CSV), tak binárních (např. HDF).

Textové soubory

- Textový soubor je posloupnost znaků členěná na řádky

Zpravidla členěná na řádky. Není to nutné, ale zvyšuje čitelnost a usnadňuje zpracování souboru (po řádcích).
- EOL (End of Line) – znak konce řádku
- EOL je platformově závislý
 - CR – Carriage Return – Macintosh – `"\r"` – 0x0d
 - LF – Line Feed – Unix – `"\n"` – 0x0a
 - CR/LF – MS-DOS, Windows – `"\r\n"` – 0x0d 0x0a
- Každý znak je reprezentován jedním bajtem, případně 2 nebo více bajty

Viz znakové sady a kódování

Binární soubory

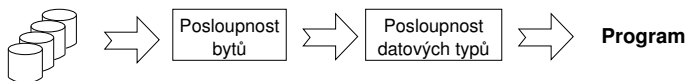
- Binární soubor je posloupnost bajtů
- Informace v binárním souboru je kódována vnitřním kódem počítače
- Do binárního souboru mohou být zapsány
 - bajt (byte)
 - jednoduché proměnné
 - pole
 - data celých objektů

V Javě lze využít tzv. [serializace](#)

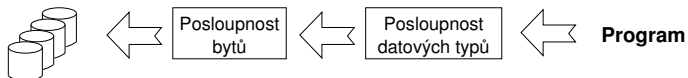
Informace o typu souboru ani o způsobu kódování informací v něm uložených není v souboru obsažena. Správnou interpretaci přečteného souboru musí zajistit uživatelský program.

Přístup k souborům

- Přenos informace (dat) z/do souboru lze rozdělit do několika vrstev
- Vrstva může poskytovat různý pohled na obsah souboru
- V základním pohledu je každý soubor **posloupnost bytů**
- Čtení ze souboru



- Zápis do souboru



Výhoda vrstveného přístupu je v možnosti jednoduše přidávat nové způsoby zpracování dat.

- K datům v souboru můžeme přistupovat dvěma základními způsoby: **sekvenčně** a **přímým** (náhodným) přístupem

Sekvenční přístup

- Při sekvenčním přístupu jsou jednotlivé byty načítány postupně
- Během načítání bytů mohou být data postupně interpretovaná
 - Např. po přečtení 4 bytů je možné interpretovat takovou posloupnost jako celé číslo typu `int`.*
- Na aktuální pozici v souboru ukazuje tzv. **kurzor**
- Každé další čtení ze souboru vrací příslušný počet přečtených bytů a o stejný počet bytů je kurzor posunut
- Při načítání se lze “vracet” pouze na začátek, nelze se vrátit např. o několik bytů zpět
- Při zápisu jsou postupně ukládány další byty na konec souboru
 - Při otevření souboru rozlišujeme kromě otevření pro čtení také otevření pro zápis nebo přidávání (`append`).*
- Sekvenční přístup načítání / zápisu je možné použít i pro jiné vstupy/výstupy než soubory uložené na disku
 - Např. Zpracování dat po sériovém portu, Ethernet nebo obecně data z Internetu*

Přímý přístup

- Při práci se soubory v přímém přístupu je možné zapisovat / číst na libovolné místo v souboru
- Práce se souborem se tak podobá přístupu k položkám v poli
- **Kurzor** lze libovolně nastavovat v rozsahu velikosti souboru (v bytech)
- Soubor musí být k dispozici

Vhodné pro soubory, které jsou přístupné na disku, a které lze "celé" kdykoliv načíst do paměti.
- Vhodné pokud známe vnitřní strukturu souboru a můžeme se přímo odkazovat na příslušné místo pro aktualizaci nebo načtení příslušné datové položky

<https://docs.oracle.com/javase/tutorial/essential/io/rafs.html>

Soubory a proudy

- Java rozlišuje soubory („*files*”) a proudy („*streams*”)
 - **Soubor** je množina údajů uložená ve vnější paměti počítače
 - **Proud** je přístup (nástroj) k přenosu informací z/do souboru, ale také z/do libovolného jiného média, které je schopné generovat nebo pojmout data jako posloupnost bytů

sítě, sériová linka, paměť, jiný program, atd.
- Informace může mít tvar znaků, bytů, skupin bytů, objektů, ...
- Přenos informace se děje ve více vrstvách v proudech (**streams**)
 1. **Otevření** přenosového proudu pro **byty** nebo **znaky**
 2. **Otevření** přenosového proudu pro **datové typy Javy**
 3. **Filtrace** dat podle dalších požadavků, např. bufferování, řádkování, atd.

Proudy v Javě (Standardní třídy)

- **Bytové – `FileInputStream` / `FileOutputStream`**
 - **`DataOutputStream`** – přenos primitivních datových typů
 - **`ObjectOutputStream`** – přenos objektů
 - **`BufferedOutputStream`** – bufferování
- **Znakové – `FileReader` / `FileWriter`**
 - **`BufferedReader`** – bufferování
 - **`StreamTokenizer`** – tokenizace

<https://docs.oracle.com/javase/8/docs/api/java/io/StreamTokenizer.html>

- **`RandomAccessFile`** – práce se soubory s náhodným přístupem
- **`File`** – zpracování souborů/adresářů: test existence, oddělovač adresářů/souborů, vytvoření, mazání, atd.

Využívá služeb operačního systému

- V Javě jsou příslušné třídy definovány v balíku **`java.io`** případně **`java.nio`**

Příklad – Soubor jako posloupnost bytů

Vytvoření kopie vstupního souboru

- Vstupní soubor postupně načítáme byte po byte a ukládáme do výstupního souboru

```
public void demoStreamCopy(String inputFile,
    String outputFile) throws IOException {
    FileInputStream in = new FileInputStream(inputFile);
    FileOutputStream out = new FileOutputStream(
        outputFile);
    int b = in.read(); // read byte of data
    while (b != -1) {
        out.write(b);
        b = in.read();
    }
    out.close();
    in.close();
}
```

lec12/DemoFileStream.java

Příklad – DemoBufferedCopyException 1/2

- V předchozím příkladě byl soubor kopírován byte po byte
- Každá operace je provedena a celkově je kopírování velmi pomalé
- Použitím „buffered“ vrstvy kopírování výrazně zrychlíme – třídy **BufferedInputStream** a **BufferedOutputStream**

```
public void demoStreamCopy(String inputFile, String
    outputFile) throws IOException {
    BufferedInputStream in = new BufferedInputStream(new
        FileInputStream(inputFile));
    BufferedOutputStream out = new BufferedOutputStream(
        new FileOutputStream(outputFile));
    int b = in.read(); // read byte of data
    while (b != -1) {
        out.write(b);
        b = in.read();
    }
}
```


Příklad – DemoBufferedCopyException 2/2

- Další vrstva pracuje s původním proudem, který předáváme v konstruktoru
- Základní velikost vyrovnávací paměti („buffer“) je 8192 bytů
DEFAULT_BUFFER_SIZE
- Přístup k souboru probíhá hromadně
- Při zápisu je však nutné vynutit zápis voláním **flush()**
Zápis je fyzicky proveden až po naplnění vyrovnávací paměti

```
int b = in.read(); // read byte of data
while (b != -1) {
    out.write(b);
    b = in.read();
}
out.flush(); // Important! Flush the output!
```

lec12/DemoBufferedFileStream.java

Výjimky (Exceptions)

- Představují mechanismus ošetření chybových (výjimečných) stavů
Program navrhujeme a implementujeme tak, aby v rámci daných operačních vlastností fungoval správně.
- Mechanismus výjimek umožňuje se „soustředit“ na funkčnost programu a „odložit“ řešení detekované chyby provedení příkazu (metody) mimo hlavní tělo funkce (metody)
- Chyba nemusí znamenat ukončení programu – chybu je možné ošetřit, zotavit běh programu a pokračovat ve vykonávání dalšího kódu
- Posloupnost příkazů, ve které může vzniknout výjimka, uzavíráme do bloku klíčovým slovem **try**
- Příslušnou výjimku pak „zachytáváme“ prostřednictvím **catch**
- Metodu můžeme deklarovat jako metodu, která může vyvolat výjimku – klíčovým slovem **throws**
- Java ošetření výjimek vynucuje

Výjimka nikoliv vyjímka – výjimka označuje název děje nebo výsledku děje, je to podstatné jméno odvozené od slovesa.

Výjimky – Příklady

- Příklad deklarace funkce, ve které může nastat výjimka

```
public void copy(String in, String out) throws
    IOException {
    ...
}
```

- V těle funkce nemusíme ošetřovat zotavení z chyby
- Řízení programu je předáno do nadřazené funkce, kde můžeme implementovat ošetření chyby, např.

```
try {
    copy("in.txt", "out.txt");
} catch (IOException e) {
    System.err.println("Error in copy");
}
```

Výjimky a výpočetní tok

- Metoda může skončit standardně nebo může být přerušena vyvoláním výjimky
- Při vyvolání výjimky se řízení předá „nejbližší“ konstrukci ošetřující výjimku **daného typu**

Výjimky jsou objekty – instance tříd, které mají svou hierarchii (viz dědičnost a polymorfismus)

- Pokud taková konstrukce v těle funkce (metody) není, dojde k nes-tandardnímu ukončení funkce a **výjimka se šíří** do nadřazené úrovně
- Není-li výjimka ošetřena ani ve funkci **main**, program vypíše výjimku na standardní chybový výstup („stderr“) a skončí
- Výjimka typu **RuntimeException** je výjimečná výjimka, kterou není nutné explicitně ošetřovat nebo deklarovat v hlavičce metody uvedením **throws**
Typicky se jedná o situace, které lze standardně ošetřit např. větvením programu (použitím if), více viz PR2.

Příklad ošetření chybových stavů

```
try {
    demo.demoStreamCopy(args[0], args[1]);
} catch (FileNotFoundException e) {
    System.err.println("File not found");
} catch (IOException e) {
    System.err.println("Error occurred during copying");
    e.printStackTrace();
}
```

■ Příklad spuštění programu

```
java DemoCopyException in2.txt out.txt
File not found
```

```
java DemoCopyException in.txt out2.txt
Error occurred during copying
java.io.IOException: Stream Closed
    at java.io.FileOutputStream.write(Native Method)
    at java.io.FileOutputStream.write(FileOutputStream.java:295)
    at DemoCopyException.demoStreamCopy(DemoCopyException.java:16)
    at DemoCopyException.main(DemoCopyException.java:24)
```

Proč jsou chyby různé?

Příklad – DemoCopyException

```
public void demoStreamCopy(String inputFile, String
    outputFile) throws IOException {
    FileInputStream in = new FileInputStream(inputFile);
    FileOutputStream out = new FileOutputStream(outputFile);
    if (outputFile.equalsIgnoreCase("out2.txt")) {
        out.close();
    }
    int b = in.read(); // read byte of data
    while (b != -1) {
        out.write(b);
        b = in.read();
    }
}
```

lec12/DemoCopyException.java

Soubor jako posloupnost primitivních typů – Zápis

- Pro zápis hodnoty základního datového typu jako posloupnost bytů přidáme další vrstvu **DataOutputStream**

```
String fname = args.length > 0 ? args[0] : "out.bin";
```

```
DataOutputStream out = new DataOutputStream(  
    new FileOutputStream(fname));
```

```
for (int i = 0; i < 10; ++i) {  
    double d = (Math.random() % 100) / 10.0;  
    out.writeInt(i);  
    out.writeDouble(d);  
    System.out.println("Write " + i + " " + d);  
}
```

```
lec12/DemoFilePrimitiveTypesWrite.java
```

Soubor jako posloupnost primitivních typů – Čtení

```
String fname = args.length > 0 ? args[0] : "out.bin";
DataInputStream in = new DataInputStream(
    new FileInputStream(fname));

for (int i = 0; i < 10; ++i) {
    int v = in.readInt();
    double d = in.readDouble();
    System.out.println("Read " + v + " " + d);
}
```

lec12/DemoFilePrimitiveTypesRead.java

Co se stane když zaměníme pořadí načítání `readInt` a `readDouble`?

Soubor primitivních typů a objektů

- Uvedenými metodami lze zapisovat a číst pouze tzv. **serializovatelné objekty**, mezi které patří
 - Primitivní datové typy
 - Řetězce a pole primitivních typů
 - Složitější objekty, pokud implementují rozhraní **Serializable**
- Rozhraní **Serializable** nepředefisuje žádnou metodu, je značkou, že objekt chceme serializovat

Pro vytvoření příslušné implementace pro převod hodnot do/z posloupnosti bytů.
- Pro serializaci musí být každá datová položka serializovatelná
- nebo označena, že nebude serializována klíčovým slovem **transient**

<https://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html>

Informativní

Příklad serializace 1/3

```
import java.io.Serializable;  
  
public class Customer implements Serializable {  
  
    private String name;  
    private String surname;  
    private int age;  
  
    public Customer(String name, String surname, int age) {  
        this.name = name;  
        this.surname = surname;  
        this.age = age;  
    }  
}
```

lec12/Customer.java

Příklad serializace 2/3

```
void write(Customer customer, String fname) throws
    IOException {
    try (ObjectOutputStream out = new
        ObjectOutputStream(new FileOutputStream(fname))) {
        out.writeObject(customer);
    }
}
```

```
Customer read(String fname) throws IOException,
    ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(new
        FileInputStream(fname));
    return (Customer) in.readObject();
}
```

lec12/DemoObjectSerialization.java

Příklad serializace 3/3

```
Customer customer = new Customer("AAA", "BBB", 47);
System.out.println("Customer: " + customer);
write(customer, fname);
customer = new Customer("ZZZ", "WWW", 17);
System.out.println("Customer: " + customer);
customer = read(fname);
System.out.println("Customer: " + customer);
```

■ Příklad výstupu

```
Customer: AAA BBB age: 47
Customer: ZZZ WWW age: 17
Customer: AAA BBB age: 47
```

Soubory s náhodným přístupem 1/2

- Třída **RandomAccessFile** pro zápis/čtení do/z libovolného místa v souboru

```
public void write(String fname, int n) throws
    IOException {

    RandomAccessFile out =
        new RandomAccessFile(fname, "rw");

    for (int i = 0; i < n; ++i) {
        out.writeInt(i);
        System.out.println("write: " + i);
    }
    out.close();
}
```

Soubory s náhodným přístupem 2/2

- Pro přístup na konkrétní položku je nutné určit „adresu“ položky v souboru jako pozici v počtu bytů od začátku souboru

```
final int SIZE = Integer.SIZE / 8;
```

```
RandomAccessFile in =  
    new RandomAccessFile(fname, "r");
```

```
for (int i = 0; i < 5; ++i) {  
    in.seek(i * 2 * SIZE);  
    int v = in.readInt();  
    System.out.println("read: " + v);  
}
```

lec12/DemoRandomAccess.java

Textově orientované soubory

- Při čtení a zápisu je nutné zajistit konverzi znaků

Kódování

- Příklad zápisu s využitím třídy `PrintWriter`

```
public void write(String fname) throws IOException {
    String months[] = {"jan", "feb", "mar", "apr", "may",
        "jun", "jul", "aug", "sep", "oct", "nov", "dec"};
};

PrintWriter out = new PrintWriter(fname, "UTF-8");
for (int i = 0; i < months.length; ++i) {
    out.println(months[i]);
}
out.close();
}
```

Příklad čtení textového souboru

- Pro čtení můžeme využít třídy **Scanner** podobně jako při čtení ze standardního vstupu

```
public void start() throws IOException {
    String fname = "text_file.txt";

    write(fname);

    FileInputStream in = new FileInputStream(fname);
    Scanner scan = new Scanner(in);
    while (scan.hasNext()) {
        String str = scan.next();
        System.out.println("Read: " + str);
    }
    in.close();
}
lec12/DemoTextFile.java
```


Shrnutí přednášky

Diskutovaná témata

- Soubory a přístup k souborům
- Typy souborů (textový a binární)
- Práce se soubory v Javě
 - Binární soubory
 - Textové soubory
- Ošetření chyb mechanismem výjimek