UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Aplikace tříd souvisejících s přenosem dat mezi operační pamětí a diskem

Marek Jelínek

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Marek Jelínek**
Osobní číslo:	**I10078**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace tříd souvisejících s přenosem dat mezi operační pamětí a diskem**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem je návrh a implementace instruktážní aplikace zaměřené na názornou demonstraci konstruktorů a metod tříd jazyka Java určených na přenos dat mezi OP a přídavnými zařízeními
(třídy File, třídy pro práci s datovými proudy a pod.). Vytvořená aplikace má sloužit jako pomůcka při výuce předmětů používajících jazyk Java.
Seznámit se na potřebné úrovni s třídami pracujícími s přenosem dat (File, InputStream, OutputStream, Reader, Writer atd...).
Zvládnout práci v grafickém režimu jazyka Java.
Vytvořit prostředí v GUI, do kterého budou implementovány instruktážní programy.
Do tohoto prostředí aplikovat množinu instruktážních programů pro jednotlivé třídy.
Součástí práce budou ukázky zdrojových textů a textové informace o vykonávaných činnostech.
Činnosti instruktážních programů a jejich další vlastnosti podle požadavků vedoucího práce...

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Herout, P.: Učebnice jazyka Java, Koop, České Budějovice, 2001
Herout, P.: Java - Grafické uživatelské prostředí a čeština, Koop, České Budějovice, 2001

Vedoucí bakalářské práce: **RNDr. Miroslav Benedikovič**
Katedra softwarových technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 10. 8. 2013 Marek Jelínek

Poděkování

Tímto bych rád poděkoval rodičům za podporu během celého studia hlavně po finanční stránce. Dále všem, kteří mě psychicky podporovali a drželi v těžkých chvílích studia. V neposlední řadě bych chtěl poděkovat mému vedoucímu práce panu RNDr. Miroslavu Benedikovičovi za cenné rady při vývoji aplikace.

Anotace

Bakalářská práce se zabývá tvorbou aplikace, která slouží jako pomůcka při výuce, kde uživatel může odzkoušet nebo nastudovat funkčnost z popisu jednotlivých metod, konstruktorů a tříd zabývajících se přenosem dat.

Klíčová slova

Java, proud, soubor, výuková aplikace

Title

Application classes associated with data transfer between memory and disk.

Annotation

This Bachelor thesis deals with creating an application that serves as a teaching tool. In this application user can test or study the functionality from the description of each methods, constructors, and classes dealing with the data transfer.

Keywords

Java, stream, file, educational application

Obsah

Seznam zkratek

ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
GUI	Graphical User Interface
URI	Uniform Resource Identifier

Seznam obrázků

Seznam tabulek

Úvod

Při programování různých aplikací se neobejdeme bez ukládání dat na disk a práce se soubory nebo adresáři. Proto je náplní práce návrh aplikace, která se bude zabývat přenosem dat mezi operační pamětí a diskem. Celý program je navržen jako prostředek pro podporu výuky jazyku Java. Budou zde implementovány jednotlivé třídy pro práci s proudy a soubory.

V textové části práce bude uveden popis jednotlivých proudů, které jsou určeny pro přenos dat. Dále bude popsána třída File pro práci se soubory a adresáři, kde bude objasněna funkčnost jednotlivých metod a práce s touto třídou a lehce se dotkne práce s formátovacími metodami a skenováním.

U praktické části bude popsán vzhled a práce s aplikací. Ta bude vytvořena tak, aby pro uživatele byla co nejpřehlednější a nejjednodušší na ovládání. Program bude implementovat několik tříd, kde uživatel bude moci prakticky vyzkoušet jednotlivé metody vybrané třídy. Dále uživatel bude mít u jednotlivých tříd, konstruktorů a metod k dispozici popis funkčnosti a příklad použití.

Aplikace bude využívat objektového programování, dále programování proti rozhraní a v neposlední řadě bude odchytávat výjimky tak, aby nedocházelo k neočekávanému pádu celého programu.

Pro pochopení textu této práce, je předpoklad základní znalosti jazyku Java.

1 Streamy

1.1 Úvod do streamů

Vstupně-výstupní proud představuje vstupní zdroj nebo výstupní cíl. Stream může zastupovat mnoho různých typů zdrojů a cílů, včetně diskových souborů, zařízení, jiných programů a paměťových polí.

Proudy jsou kompatibilní s mnoha odlišnými typy dat, včetně jednoduchých bajtů, základních datových typů, lokalizovaných znaků a objektů. Některé proudy jednoduše předávají data. Jiné s daty manipulují a užitečným způsobem modifikují. [1]

Java pro práci se vstupy a výstupy poskytuje celou řadu tříd a jejich metod. Ty jsou uloženy v balíku `java.io`[1]. Knihovna je založena na mechanizmu tzv. vstupních a výstupních proudů (streamů).

Stream si lze představit jako trubku, kde je k dispozici její konec. Z něj lze data číst nebo naopak zapisovat. Java poskytuje celou řadu streamů, z pohledu uživatele mají většinou stejnou funkcionalitu. Java obsahuje čtyři základní třídy pro tvorbu proudů. Pokud je třeba něco speciálního, lze vhodnou třídu vytvořit (resp. odvodit od nějaké existující). Streamy se dělí na dvě hlavní kategorie, bajtové a znakové. Rozdíl je ve způsobu práce se znaky. Třída `InputStream`[2] je základní abstraktní třída pro bajtově orientovaný vstupní proud. Třída `OutputStream`[3] slouží pro bajtově orientovaný výstup. Pro znakové vstupy slouží třída `Reader`[4]. Třída `Writer`[5] se využívá u znakových výstupů. Mimo třídy pro proudy balík `java.io` obsahuje také třídu `File`[6], která slouží pro práci se soubory a adresáři.

1.2 Druhy streamů

Existuje mnoho druhů vstupně-výstupních proudů. Ty pracují prakticky stejně, ať se jedná o data v souborech na disku, o síťovou komunikaci, komunikaci mezi vlákny apod. Existují ovšem podstatné rozdíly v přípravě streamů před komunikací. [2][3]

1.2.1 Bajtové proudy

Programy pomocí bajtových proudů zajišťují vstup a výstup 8bitových bajtů. To znamená, že každý znak je zapisován (resp. čten), postupně za pomoci nějakého cyklu. Všechny třídy bajtových proudů jsou potomky tříd `InputStream` a `OutputStream`. Pro práci se soubory se používají třídy `FileInputStream`[7] a `FileOutputStream`[8]. Jiné typy proudů se používají podobně, pouze se liší stylem svého návrhu.

[1] http://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html
[2] http://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html
[3] http://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html
[4] http://docs.oracle.com/javase/7/docs/api/java/io/Reader.html
[5] http://docs.oracle.com/javase/7/docs/api/java/io/Writer.html
[6] http://docs.oracle.com/javase/7/docs/api/java/io/File.html
[7] http://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html
[8] http://docs.oracle.com/javase/7/docs/api/java/io/FileOutputStream.html

Pokud soubor, ze kterého čteme, obsahuje znaková data, je lepší zvolit znakové proudy. Důvodem je, že bajtový proud čte (resp. zapisuje) po jednotlivých bajtech. To je ovšem pomalé. Proto jsou vhodné k nejjednodušším vstupně-výstupním operacím.

1.2.2 Znakové proudy

Znakové proudy fungují stejným způsobem jako ty bajtové, pouze operují s textem. To má za následek zrychlení oproti bajtovým. Všechny třídy znakových proudů jsou potomky tříd `Reader` a `Writer`. Pro práci se soubory jsou zde třídy `FileReader`[9] a `FileWriter`[10].

Java ukládá znakové hodnoty pomocí znakové sady Unicode. Proto automaticky převádí tento vnitřní formát na místní znakovou sadu a zpět. Místní znakovou sadou se obvykle myslí 8bitová nadmnožina sady ASCII.

Znakové proudy často fungují jako obálky bajtových proudů. Znakový proud zajišťuje fyzický vstup a výstup pomocí bajtového proudu, zatímco samotný znakový proud řeší převod mezi znaky na bajty. Třída `FileReader` například používá třídu `FileInputStream` zatímco třída `FileWriter` pracuje se třídou `FileOutputStream`.

1.2.3 Standardní proudy

Standardní proudy čtou vstup z klávesnice a zapisují výstup na zobrazovací zařízení. Obsahují i vstupně-výstupní operace pro práci se soubory a mezi programy, tuto funkci řídí interpret příkazového řádu, nikoli vlastní program.

Existují tří standardní proudy. Pro vstup se využívá objekt `System.in`, výstup je přístupný přes objekt `System.out` a standardní chybový výstup na základě objektu `System.err`. Všechny tři jsou definovány automaticky a není nutné je otevřít.

Jedná se o bajtové proudy, kde `System.out` a `System.err` jsou definovány jako objekty typu `PrintStream`[11]. Ten je bajtový, ale pomocí vnitřního objektu znakových proudů emuluje mnoho funkcí. `System.in` je bajtový proud, který neobsahuje funkce znakových proudů. Je možné zabalit `System.in` do objektu `InputStreamReader`[12]. Poté lze standardní vstup použít jako znakový proud.

```
InputStreamReader cin = new InputStreamReader(System.in);
```

1.2.4 Datové proudy

Datové proudy umožňují binární vstup a výstup hodnot základních datových typů (`boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double` a `String`). Všechny datové

[9] http://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html
[10] http://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html
[11] http://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html
[12] http://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html

proudy implementují rozhraní `DataInput`[13] nebo `DataOutput`[14]. Nejčastěji používané implementace těchto rozhraní jsou třídy `DataInputStream`[15] a `DataOutputStream`[16].

1.2.5 Objektové proudy

Pomocí objektových proudů lze zajistit vstup a výstup objektů. Většina standardních tříd podporuje serializaci objektů. U nich je implementováno typické rozhraní `Serializable`[17].

Objektové proudy jsou instancemi tříd `ObjectInputStream`[18] a `ObjectOutputStream`[19]. Ty implementují rozhraní `ObjectInput`[20] a `ObjectOutput`[21]. Jelikož jsou tato rozhraní podřízena `DataInput` a `DataOutput`, všechny vstupně-výstupní metody základních dat popsané v oddílu Datové proudy, jsou implementovány i v objektových proudech. Proto může obsahovat směs základních a objektových hodnot. [1]

1.2.6 Buffered Stream

Třídy `BufferedInputStream`[22] a `BufferedOutputSteam`[23] obsahují pole, které slouží jako vyrovnávací paměť. Pokud čteme z disku, proud načte celý blok dat a uloží jej do pole. Poté data posílá dále programu. Jakmile se pole vyprázdní, učiní nový dotaz na další blok. Tento způsob značně urychlí proces čtení (resp. zapisování).

Při zápisu na disk může nastat situace, kdy zapíšeme do bufferu méně dat, než je potřeba k jeho vyprázdnění. V tomto případě lze zavolat metodu `flush()`, která vyprázdní všechny buffery (okamžitý zápis dat). Metoda `close()`, která slouží pro uzavírání proudů, automaticky vyprázdní všechny buffery, není tedy třeba volat `flush()`. [4]

1.3 Životní cyklus proudu

Každý stream má svůj životní cyklus. Ten je velmi jednoduchý.

1. Vytvoření, zavolání konstruktoru - může se vytvářet přímo tam, kde se používá nebo ho může vytvořit jiný objekt.
2. Otevření - většinou se stream otevírá už při vytvoření, pokud nebyl otevřen, musí být později otevřen, jinak s ním nelze pracovat. Při otevření se alokují potřebné systémové prostředky, které připraví stream na práci.
3. Vlastní práce - zde se volají příslušné metody a ty provádějí požadované operace

[13] http://docs.oracle.com/javase/7/docs/api/java/io/DataInput.html
[14] http://docs.oracle.com/javase/7/docs/api/java/io/DataOutput.html
[15] http://docs.oracle.com/javase/7/docs/api/java/io/DataInputStream.html
[16] http://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html
[17] http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html
[18] http://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html
[19] http://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html
[20] http://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html
[21] http://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutput.html
[22] http://docs.oracle.com/javase/7/docs/api/java/io/BufferedInputStream.html
[23] http://docs.oracle.com/javase/7/docs/api/java/io/BufferedOutputStream.html

4. Uzavření - poslední etapa cyklu je velmi důležitá. Pokud program přestane využívat daný proud, je třeba jej vždy uzavřít. Korektně finalizovaný objekt streamu nebo normálně ukončený program zaručuje správné zavření, ale nelze na to spoléhat. Neukončený proud vyčerpává systémové prostředky a při zápisu může mnoho dat zůstat nezapsaných.

1.4 Ošetření chyb

Důležité při práci se streamy je ošetření chyb, které se mohou vyskytnout. Vyskytne-li se nějaký chybový stav, většinou je řešený výjimkami. Většina streamových metod vyvolá výjimku `IOException`[24], která je synchronní, a proto musí být deklarována. Zahrnuje do sebe celou škálu podtříd a je společná všem streamům bez ohledu na implementaci. Proudy mohou obsahovat i jiné výjimky. [2]

1.5 Serializace a deserializace

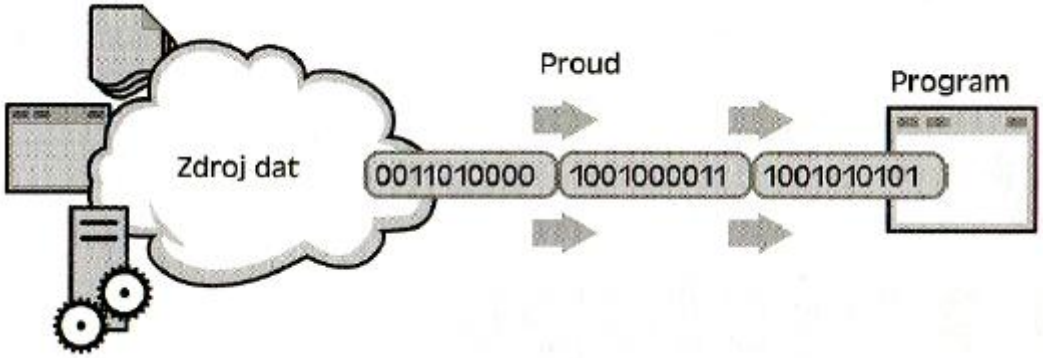
Často je zapotřebí uložit nebo přenést primitivní nebo složitější datové typy na jiné místo. Musí se to provést tak, aby se v jiném čase nebo na jiném místě data správně zrekonstruovala do původní podoby. Tyto činnosti se nazývají serializace a deserializace.

Při serializaci objektů jsou k dispozici třídy `ObjectInputStream` a `ObjectOutputStream`. Ty nejenže ukládají a načítají instance objektů, ale poradí si i s primitivními typy. Nelze ukládat všechny objekty. Nutnou podmínkou je, aby implementovaly rozhraní `Serializable`. Další podmínka, kterou je nutno splnit, je, že každá nadtřída, která neimplementuje `Serializabel`, musí mít veřejný bezparametrický konstruktor. Serializovaná třída musí zajistit serializaci datových složek všech nadtříd, které nejsou serializovatelné. Protože se instance serializuje i se všemi odkazovanými objekty, musí být i tyto serializovatelné, anebo označené modifikátorem `transient` (tedy že nebudou uloženy). Na rozdíl od primitivních typů, u objektů lze při deserializace zjistit jejich typ. [4][6]

1.6 Vstupní proud

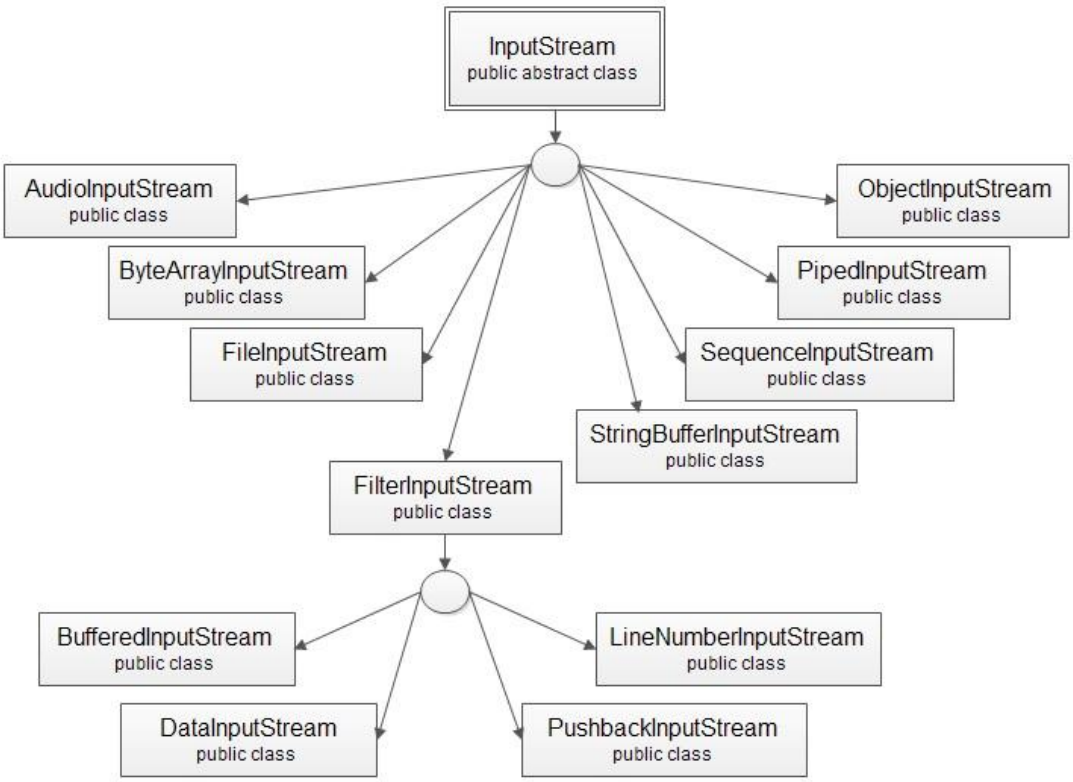
Program pomocí vstupního proudu čte data ze zdroje po jednotlivých položkách (Obrázek 1).

[24] http://docs.oracle.com/javase/7/docs/api/java/io/IOException.html



Obrázek 1 - Načítání informací do programu

Pro vstupy se používají proudy založené na třídě `InputStream` a `Reader`. Na obrázku 2 je struktura třídy `InputStream`. Struktura dědičnosti třídy `Reader` je obdobná.



Obrázek 2 - Struktura dědičnosti třídy InputStream

Třídy pro vstup lze rozdělit do čtyř samostatných skupin, třídy pro vytvoření vstupního proudu, třídy poskytující další vlastnosti vstupnímu proudu, třídy pro vytvoření readeru a třídy pro rozšíření funkcí readeru. V následujících tabulkách je uveden jejich přehled.

Tabulka 1 - Třídy pro vstup po bytech

Třída	Použití
InputStream	abstraktní třída, která definuje základní metody pro čtení po bytech
AudioInputStream	třída pro práci se zvukem
ByteArrayInputStream	čtení z pole bajtů v paměti, které je parametrem konstruktoru
FileInputStream	čtení ze souboru, parametrem konstruktoru je `String` se jménem souboru nebo objet typu `File`
ObjectInputStream	deserializuje primitivní data a objekty, které byly dříve zapsané pomocí `ObjectOutputStream`
PipedInputStream	čtení z roury (z objektu, do kterého zapisuje `PipedOutputStream`)
SequenceInputStream	vytvoří jeden vstupní proud ze dvou vstupních proudů, které jsou parametrem konstruktoru
StringBufferInputStream	převede `String` na `InputStream`, doporučuje se používat místo toho `StringReader`

Tabulka 2 - Třídy přidávající funkčnost pro čtení po bytech

Třída	Použití
FilterInputStream	předek tříd, které čtou z jiného InputStreamu a přidávají k tomu další funkčnost
BufferedInputStream	vytváří buffer pro čtení, čímž je toto čtení efektivnější
DataInputStream	ve spolupráci s třídou `DataOutputStream` umožňuje přenášet údaje ve formátu přenositelném mezi různými platformami
LineNumberInputStream	přidává metodu pro číslování čtených řádků, doporučuje se použít `LineNumberReader`
PushbackInputStream	umožňuje vrátit část přečtených bajtů zpět do vstupního proudu

Pro čtení po znacích je k dispozici třída `Reader` a její potomci. Měli by se používat vždy, když se čte text, neboť v této třídě je garantována správná obsluha znakových sad a převod textu do vnitřního kódování Javy. V následující tabulce je přehled tříd vytvoření readeru a jejich srovnání s potomky třídy `InputStream`.

Tabulka 3 - Třídy pro čtení po znacích a jejich obdoba pro čtení po bytech

Třída	Použití	Odpovídající InputStream
Reader	abstraktní třída, která definuje základní metody pro čtení po znacích	`InputStream`
InputStreamReader	převádí `InputStream` na `Reader`	-
FileReader	čtení ze souboru, parametrem	`FileInputStream`

	konstruktoru je `String` se jménem souboru nebo objekt typu `File`	
PipedReader	čtení z roury (z objektu do kterého zapisuje `PipedWriter`)	`PipedInputStream`
CharArrayReader	čtení z pole znaků v paměti, které je parametrem konstruktoru	`ByteArrayInputStream`
StringReader	převede `String` na `Reader`	`StringBufferInputStream`

Tabulka 4 - Třídy pro rozšíření funkčnosti readeru

Třída	Použití	Odpovídající InputStream
FilterReader	předek tříd, které čtou z jiného Readeru a přidávají k tomu další funkčnost	`FilterInputStream`
BufferedReader	vytváří buffer pro čtení, současně přidává metodu `ReadLine()` pro čtení po řádcích	`BufferedInputStream`
LineNumberReader	přidává metodu pro číslování čtených řádků	`LineNumberInputStream`
PushbackReader	umožňuje vrátit část přečtených znaků zpět do vstupního streamu	`PushbackInputStream`

1.6.1 Čtení ze souboru

Nejčastěji používaným vstupem je textový soubor. Pro čtení po znacích je třeba vytvořit instanci třídy `FileReader`.

```
FileReader fr = new FileReader("soubot.txt");
```

Poté lze číst znaky z tohoto souboru pomocí metody `read()`.

```
int zn = fr.read();
```

Lepší variantou je ovšem čtení po řádcích. Instance `FileReader` se musí zabalit do filtru `BufferedReader`.

```
BufferedReader radek = new BufferedReader(fr);
```

Poté použít metodu `readLine()`.

```
String str = radek.readLine();
```

Obě tyto třídy poskytují metodu `close()`, která se postará o zavření souboru. Pro testování konce souboru se u čtení po bytech využívá hodnota -1 a při čtení po znacích hodnota null.

Příklad čtení ze souboru.

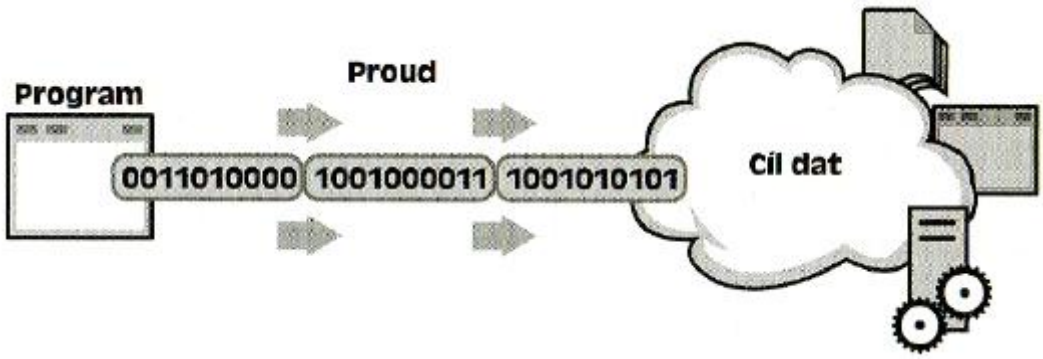
```
import java.io.*;
public class CteniZeSouboru {
    public static void main(String[] args) {
        try{
```

```
            BufferedReader vstup = new BufferedReader
                                   (new FileReader("soubor.txt"));
            String s;
            while((s = vstup.readLine()) != null)
                System.out.println(s);
            vstup.close();
        } catch(IOException e) {
            System.out.println("chyba na vstupu souboru");
        }
    }
}
```

Výjimka `IOException` musí být odchycena. Bez ošetření není tento program přeložitelný.

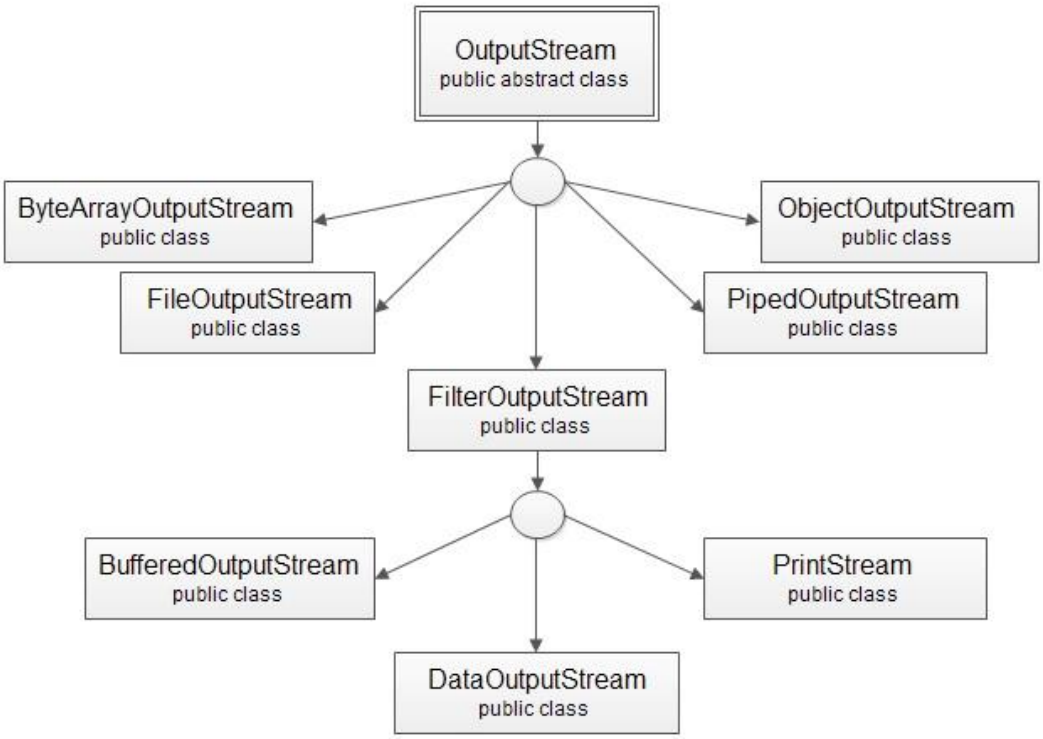
1.7 Výstupní proud

Výstupní proud programu umožňuje zapisovat data na cíl po jednotlivých položkách (Obrázek 3).



Obrázek 3 - Zapisování informací z programu

Pro výstupy se používají proudy založené na třídě `OutputStream` a `Writer`. Na obrázku 4 je struktura třídy `OutputStream`. Struktura dědičnosti třídy `Writer` je obdobná.



Obrázek 4 - Struktura dědičnosti třídy OutputStream

Stejně jako u vstupu lze třídy pro výstup rozdělit do čtyř skupin. Třídy pro vytvoření vstupního proudu, třídy pro rozšíření funkčnosti výstupního proudu, třídy pro vytvoření writeru a třídy pro rozšíření funkčnosti writeru.

Tabulka 5 - Třídy pro zápis po bytech

Třída	použití
Outputstream	abstraktní třída, která definuje základní metody pro zápis po bytech
FileOutputStream	zápis do souboru, parametrem konstruktoru je `String` se jménem nebo objekt typu `File`
ByteArrayOutputStream	zápis do roury (do objektu, ze kterého čte `PipedInputStream`)
PipedOutputStream	zápis do pole bytů v paměti, které je parametrem konstruktoru

Tabulka 6 - Třídy pro přidání funkčnosti při zápisu po bytech

Třída	použití
FilterOutputStream	předek tříd, které přidávají k Outputstreamu funkčnost
BufferedOutputStream	vytváří buffer pro efektivnější zápis

DataOutputStream	ve spolupráci s třídou `DataInputStream` umožňuje přenášet údaje ve formátu přenositelném mezi různými platformami
PrintStream	formátuje výstup, poskytuje metody `print()` a `println()`

Pro zápis po znacích je k dispozici třída `Writer` a její potomci. Měli by se používat vždy, když se text zapisuje, neboť v této třídě je garantována správná obsluha znakových sad a převod textu do vnitřního kódování Javy. V následující tabulce je přehled tříd vytvoření writeru a jejich srovnání s potomky třídy `OutputStream`.

Tabulka 7 - Třída pro zápis po znacích

Třída	Použití	Odpovídající OutputStream
Writer	abstraktní třída, která definuje základní metody pro zápis po znacích	`OutputStream`
OutputStreamWriter	převádí `OutputStream` na `Writer`	-
FileWriter	zápis do souboru, parametrem konstruktoru je `String` se jménem souboru nebo objekt typu `File`	`FileOutputStream`
PipedWriter	zápis do roury (do objektu ze kterého čte `PiperReader`)	`PipedOutputStream`
StringWriter	zápis do bufferu, který může být převede do objektu `String` či `StringBuffer`	-
CharArrayWriter	zápis do pole znaků v paměti	`ByteArrayOutputStream`

Tabulka 8 - Třídy pro rozšíření funkčnosti při zápisu po znacích

Třída	Použití	Odpovídající OutputStream
FileWriter	předek tříd, které přidávají k OutputStreamu další funkčnost	`FilterOutputStream`
BufferedWriter	vytváří buffer pro efektivnější zápis	`BufferedOutputStream`
PrintWriter	formátuje výstup, poskytuje metody `print()` a `println()`	`PrintStream`

1.7.1 Zápis do textového souboru

Pro zápis do souboru se používá znakový proud. Je potřeba vytvořit nový výstupní writer. Pokud soubor na disku neexistuje, bude vytvořen nový, existuje-li, bude přepsán. Jestliže je potřeba do existujícího souboru připisovat nakonec, musí se vytvořit konstruktor třídy `FileWriter`, který má dva parametry. První je soubor, do kterého se má zapisovat. Druhý je logická hodnota, která určuje, zda se bude zapisovat na konec souboru nebo původní soubor přepisovat. Hodnota true znamená dopisování na konec.

`FileWriter` se používá pro zápis po znacích.

```
FileWriter fw = new FileWriter("soubor.txt");
```

Výhodnější je zapisovat po řádcích a proto se musí použít filtr `PrintWriter`, který má metodu `println()` pro zápis celého řádku.

```
PrintWriter vystup = new PrintWriter(fw);
```

Příklad zápisu do souboru.

```
import java.io.*;
public class ZapisDoSouboru {
    public static void main(String[] args) {
        try{
             PrintWriter vystup = new PrintWriter
                                  (new FileWriter("soubor.txt"));
             for(int i = 1; i < 11 ; i++)
                vystup.println("radek " + i);
             vystup.close();
        } catch(IOException e) {
            System.out.println("chyba pri zapisu");
        }
    }
}
```

Po skončení zápisu je opět nutné použít metodu `close()` a ošetřit výjimky. [3]

2 Skenování a formátování

Při programování vstupu a výstupu je často nutné zajistit převod na přehledně formátovaná data, která jsou čitelná pro uživatele, a převod opačným směrem. Platforma Java programátorům tuto rutinní činnost usnadňuje pomocí dvou rozhraní API. Skenovací rozhraní API rozloží vstup na jednotlivé tokeny, které jsou přidruženy k datovým prvkům. Formátovací rozhraní API sestaví data do přehledně formátovaného tvaru srozumitelného pro uživatele.

2.1 Skenování

K rozložení formátovaného vstupu na tokeny a překladu jednotlivých tokenů v závislosti na jejich datových typech se hodí objekt typu `Scanner`[25].

Ve výchozím nastavení skener odděluje tokeny pomocí prázdných znaků. Vyvoláním metody `useDelimiter()` a zadáním regulárního výrazu můžeme nastavit jiný oddělovač tokenů. Například můžeme jako oddělovač tokenů zvolit čárku, případně následovanou prázdným znakem.

```
s.useDelimiter(",\\s*");
```

Následující program ukáže, jak skenování funguje.

```
import java.io.*;
import java.util.Scanner;

public class Sken {

    public static void main(String[] args) {
        Scanner s = null;
        try{
            s = new Scanner(new BufferedReader
                                (new FileReader("soubor.txt")));
            while(s.hasNext()){
                System.out.println(s.next());
            }
        } finally {
            if(s != null) {
                s.close();
            }
        }
    }
}
```

Skener rozloží soubor na jednotlivé tokeny pomocí prázdného znaku a vypíše na výstupu jednotlivé části pod sebe.

[25] http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

2.2 Formátování

Objekty proudů, které implementují formátování, jsou instancemi třídy znakových proudů `PrintWriter` nebo třídy bajtových proudů `PrintStream`.

2.2.1 Metody print() a println()

Vyvolání metody `print()` nebo `println()` poskytne výstup jedné hodnoty, která je nejdříve převedena pomocí příslušné metody `toString()`.

Příklad použití metod.

```
public class Root {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.print("Druhá odmocnina z ");
        System.out.print(i);
        System.out.print(" je ");
        System.out.print(r);
        System.out.println(".");

        i = 5;
        r = Math.sqrt(i);
        System.out.println("Druhá odmocnina z " + i + " je " + r + ".");
    }
}
```

Program Root dává tento výsledek:

```
Druhá odmocnina z 2 je 1.4142135623730951.
Druhá odmocnina z 5 je 2.23606797749979.
```

Proměnné `i` a `r` jsou poprvé formátovány v přetížené metodě `print()` a podruhé pomocí konverzního kódu, který je automaticky generován překladačem Java. Tímto způsobem lze formátovat libovolnou hodnotu, ale pouze s omezenou kontrolou nad výsledkem.

2.2.2 Metoda format()

Metoda `format()` formátuje více argumentů v závislosti na formátovacím řetězci. Ten zahrnuje statický text proložený specifikátory formátu. Formátovací řetězec se při výstupu nemění s výjimkou specifikátoru formátu.

Všechny specifikátory začínají symbolem % a končí konverzí z jednoho či dvou znaků, která určuje typ generovaného formátovaného výstupu.

Příklad některých konverzí:

- d - formátuje celočíselnou hodnotu jako desítkové číslo.
- f - formátuje hodnotu s plovoucí desetinou čárkou jako desítkové číslo.
- n - odešle na výstup oddělovač řádků specifický pro platformu.

- x - formátuje celé číslo jako šestnáctkovou hodnotu.
- s - formátuje libovolnou hodnotu jako řetězec.
- tB - formátuje celé číslo jako název měsíce podle místního nastavení.

K dispozici je mnoho dalších konverzí.

Příklad použití metody.

```
public class Root2 {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.format("Druhá odmocnina z %d je %f.%n", i, r);
    }
}
```

Program Root2 dává tento výsledek:

```
Druhá odmocnina z 2 je 1,414214.
```

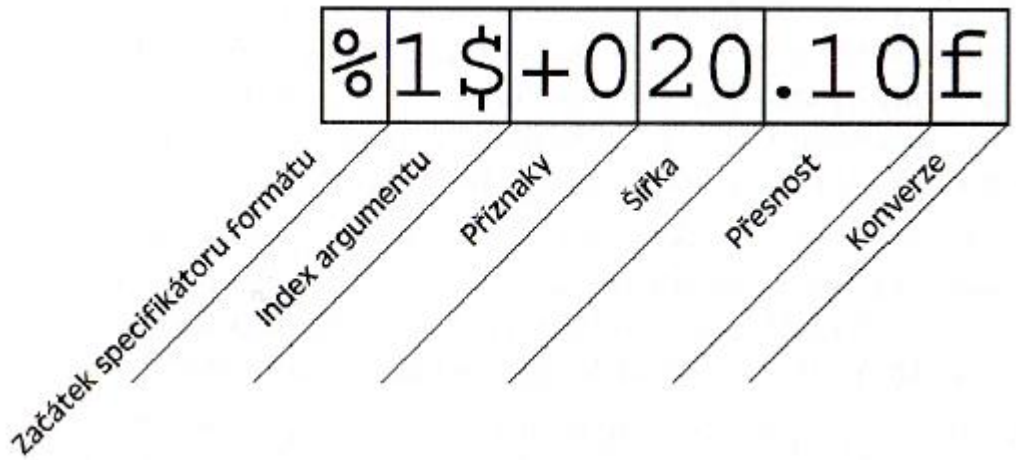
Kromě konverze může specifikátor formátu obsahovat také několik dalších prvků, které dále přizpůsobují formátovací výstup.

```
public class Format {
     public static void main(String[] args) {
         System.out.format("%f, $1$+020.10f %n", Math.PI);
     }
}
```

Program Format dává tento výsledek:

```
3.141593, +000000003.1415926536
```

Obrázek 5 znázorňuje, jak se delší specifikátor dělí na jednotlivé prvky.



Obrázek 5 - Prvky specifikátoru formátu

Prvky musí být uvedeny v zobrazeném pořadí. Směrem zprava se jedná o následující volitelné prvky:

- **Přesnost -** U hodnot s plovoucí desetinou čárkou se jedná o matematickou přesnost formátované hodnoty. V případě `s` a dalších obecných konverzí se jedná o maximální šířku formátované hodnoty, která je v případě potřeby zprava zkrácena.

- **Šířka -** Minimální šířka formátované hodnoty. Hodnota je v případě potřeby doplněna. Ve výchozím nastavení je hodnota doplněna zleva prázdnými znaky.

- **Příznaky -** Určují další možnosti formátování. V příkladu `Format` definuje příznak `+`, že číslo bude formátováno se znaménkem a příznak `0` nastaví výplňový znak na hodnotu 0. Další příznak `-` vyplnění zprava a příznak `,` formátuje čísla pomocí oddělovače tisíců dle místního nastavení. Některé příznaky nelze použít s určitými jinými příznaky nebo konverzemi.

- **Index argumentu -** Dovoluje explicitně označit vyhrazený argument. Lze také uvést symbol `<`, který označí stejný argument jako předchozí specifikátor. Příklad by mohl mít tvar: [1]

```
System.Out.format("%f, %<+020.10f %n", Math.PI);
```

3 Práce se soubory

Protože proudy nepodporují všechny operace, které jsou u diskových souborů běžné, Balíček `java.io` pro práci se soubory obsahuje třídu `File`, která umožňuje vytvářet nezávislý kód na platformě. Ten analyzuje soubory a adresáře, se kterými poté manipuluje. Samotný soubor se skoro na všech platformách chová stejně, ale způsob práce se velice liší.

Na systémech unixového typu (ext3, ReiserFS atd.) pro které je typické, že celý systém tvoří jediný adresářový strom, se jako oddělovač jednotlivých úrovní používá lomítko a rozlišují se velká a malá písmena

Systémy firmy Microsoft (FAT a NTSF) rozlišují jednotlivá logická zařízení, oddělovač je zpětné lomítko, velikost písma se nerozlišuje, kódování se liší podle jednotlivých systémů.

3.1 Třída File

Vše potřebné pro práci se soubory poskytuje třída `File`. Název této třídy je zavádějící. Instance třídy `File` nepředstavují přímo konkrétní soubor, ale tzv. abstraktní cestu. Může odkazovat na platný soubor, ale také nemusí.

V řadě tříd ze standardní knihovny Javy najdeme metody, které vyžadují jako svůj argument název souboru. Prakticky ve všech případech lze použít jak textový řetězec, tak právě instance třídy `File`, což je přenositelnější a robustnější řešení, protože můžeme už předem zjistit o daném souboru nějaké informace nebo provést se souborem potřebné operace, například přejmenování, odstranění nebo změny oprávnění.

3.1.1 Specifikace abstraktní cesty

Objekt třídy `File` může představovat jak soubory, tak adresáře. Cesta může být absolutní, relativní nebo prázdná. Abstraktní cesta se skládá z prefixu a z posloupnosti názvů jednotlivých adresářových úrovní oddělených separátorem.

Jak se s cestami pracuje, záleží na nastavení vlastností systému. Nejdůležitější je oddělovač názvů v cestě. Ve třídě `File` je určen hodnotou konstanty `separatorChar` (znakové) nebo `separator` (řetězcové). Obsahuje také konstanty `pathSeparatorChar` a `pathSeparator`. Tyto konstanty představují oddělovač cest v případě, kdy máme zapsaných několik cest za sebou a mají hodnotu dvojtečky (Unix systémy) nebo středníku (Microsoft systémy).

3.1.2 Vytvoření instance File

Jednou vytvořený objekt File už nelze měnit. Má to svou logiku, protože jestliže řetězec je v Javě neměnný, musí být jeho speciální případ také neměnný.

Objekt File lze vytvořit třemi způsoby, název souboru, název souboru vzhledem k rodiči, a pomocí URI. Pokud je vytvářen přímo z celé cesty, řetězec se pouze převede na abstraktní cestu. Je-li dán rodič (názvem nebo instancí File a není null) je abstraktní cesta vytvořena jako relativní vůči rodičovské cestě, resp. proti výchozímu adresáři, pokud

je rodič prázdná abstraktní cesta. Vytváří-li se instance File z URI, musí být splněny určité požadavky, schéma musí být file, cesta nesmí být prázdná atd.

Instance může být vytvořena například zavoláním tohoto konstruktoru.

```
File f = new File(soubor.txt);
```

Program může pomocí objektu typu File získat další verze názvu souboru. Některé z těchto názvů se mohou lišit od původního řetězce názvu souboru, který byl předán konstruktoru. Program volá různé metody, které poskytují odlišné verze názvu souboru. V tabulce 9 je uvedeno porovnání zobrazení na Windows a Linux.

Tabulka 9 - Příklad porovnání třídy File

Volaná metoda	Návratová hodnota v systému Microsoft Windows	Návratová hodnota v systému Linux
`f.toString()`	Soubor.txt	Soubor.txt
`f.getName()`	Soubor.txt	Soubor.txt
`f.getParent()`	NULL	NULL
`f.getabsolutePath()`	C:\java\examples\soubor.txt	/home/cafe/java/examples/soubor.txt
`f.getCanonicalPath()`	C:\java\examples\soubor.txt	/home/cafe/java/examples/soubor.txt

3.1.3 Operace s instancí třídy File

Protože objekt File je abstraktní cestou k souboru, lze s touto cestou pracovat a získávat různé varianty. Lze získat celou cestu v různých podobách, části a některé další verze.

- `getPath()` - vrátí abstraktní cestu v podobě, jak byla zadána při vytváření objektu. Stejný efekt má i metoda `toString()`.
- `getAboslutePath()` - vrátí cestu převedenou do absolutního tvaru. Mechanismus případného převodu z relativní cesty na absolutní je platformově závislý, většinou se ale jako báze použije domovský adresář uživatele.
- `getCanonicalPath()` - vrací kanonický tvar cesty. V praxi to znamená, že se pokusí cestu maximálně zpracovat. Odstraní všechny označení stejného nebo nadřazeného adresáře (tečku a dvě tečky), zpracuje symbolické odkazy atd. Chování je silně platformově závislé a liší se podle toho, zda cesta (nebo její části) existuje či nikoli.
- `getName()` - získá z cesty název souboru (bez adresářové cesty).
- `getParent()` - vrátí rodičovský adresář souboru. Vychází se pouze z cesty, soubor ani adresář nemusí existovat.
- `isAbsolute()` - zjistí, zda je cesta absolutní.
- `compareTo()` - lexikograficky porovná tuto abstraktní cestu s jinou (ani jedna nemusí existovat). Porovnávání je platformově závislé, podle systému se použije

rozlišení malých/velkých písmen. Podobně pracuje metoda `equals()`, která pouze zjišťuje, zda jsou abstraktní cesty totožné.

Všechny uvedené metody, které vrací cestu nebo její část, mají jako návratovou hodnotu textový řetězec. Metody `getAbsoluteFile()`, `getCanonicalFile()` a `getParentFile()` vracejí novou instanci typu `File`.

Abstraktní cestu lze také převést na odpovídající URL (`getURL()`) anebo na URI (`getURI()`)

Pro zjišťování informací o souboru jsou ve třídě file příslušné metody.

- `exists()` - zjistí zda soubor existuje. Při práci se souborem je dobré nejprve zavolat tuto metodu a ověřit jeho přítomnost.
- `isFile()`, `isDirectory()` - tyto metody zjišťují, zda se jedná o soubor nebo adresář. Je to platformově závislé, například symbolický odkaz na soubor se tváří jako běžný soubor.
- `canRead()`, `canWrite()` - metody vrátí, zda můžeme číst či zapisovat do daného souboru. Pokud byl přístup odepřen, nelze zjistit proč. Nelze zjišťovat přístupová práva.
- `isHidden()` - zjišťuje, zda je soubor označen jako skrytý. V unixových systémech skryté soubory začínají tečkou, ve Windows pak soubory s nastaveným atributem hidden.
- `length()` - zjistí velikost souboru. Protože vrací hodnotu typu `long`, není problém s velkými soubory.
- `lastModified()` - jediný časový údaj, který lze o souboru zjistit, je čas poslední modifikace. Ne všechny souborové systémy poskytují další časové informace, proto je to takto omezeno. Navíc v praxi je to právě ten nejpotřebnější údaj, podle něhož se například zjišťuje, že někdo změnil konfigurační soubor.

Pro samotné adresáře je k dispozici speciální sada metod, která se využívá pro přístup k souborům v těchto adresářích.

- `list()` - vrátí pole obsahující seznam všech souborů v daném adresáři. Položky tohoto pole budou textové řetězce s názvy souborů. Pořadí, ve kterém se soubory vypíší, není definováno, může být libovolné.
- `listFiles()` - tato metoda dělá přesně totéž co `list()`, ale místo pole textových řetězců vrací pole objektů `File`, tedy abstraktních cest.
- `list(FilenameFilter f)` - modifikace metody `list()`. Předem se určuje, které soubory se mají vybrat pomocí implementace rozhraní `FilenameFilter`.
- `listFiles(FilenameFilter f)` - metoda vracející pole objektů `File`, opět s filtrací.
- `listFiles(FileFilter f)` - modifikace, ale s jiným typem filtru.

- `listRoots()` - vrací pole všech kořenů adresářových stromů, které jsou v danou chvíli k dispozici.

Třída File nabízí několik manipulačních operací, se kterými lze něco změnit.

- `renameTo(File f)` - metoda přejmenuje soubor podle zadání.
- `delete()` - pokusí se smazat soubor. Pokud to jde, smaže ho. Neprázdné adresáře nelze mazat, musí se nejdříve vyprázdnit.
- `deleteOnExit()` - naplánuje smazání souboru při ukončení programu. Používá se pro mazání dočasných souborů.
- `createNewFile()` - Vytvoří nový prázdný soubor.
- `mkdir(), mkdirs()` - metody slouží pro vytvoření adresářů. Liší se pouze tím, že první vytvoří pouze jediný adresář, na který odkazuje instance File, kdežto druhá vytvoří i všechny nadřazené adresáře.
- `setLastModified(long time)` - změní časový údaj o poslední změně souboru.
- `setReadOnly()` - nastaví, že soubor bude pouze ke čtení. Tato operace v Javě nejde vrátit.

Občas je potřeba uložit nějaká data do dočasných souborů, aby je bylo možné použít později. Třída `File` má k dispozici dvě statické metody.

- `createTempFile(String prefix, String suffix)` - vytvoří dočasný soubor s daným prefixem (min. 3 znaky dlouhý) a danou koncovkou (může být null, pak se použije .tmp). Soubor se vytvoří v adresáři pro dočasné soubory, vrací instanci `File`.
- `createTempFile(String prefix, String suffix, File directory)` - od předchozí metody se liší tím, že umožňuje specifikovat adresář pro uložení souboru (pokud je null, chování této metody je shodně s tou předchozí).

Vytvořený soubor se nemaže automaticky při skončení programu. Pro mazání se využívá metoda `deleteOnExit()`.

3.2 Soubory s náhodným přístupem

Soubory s náhodným přístupem umožňují náhodný (ne sekvenční) přístup ke svému obsahu. Třída `java.io.RanomAccessFile`[26] implementuje rozhraní `DataInput` i `DataOutput`, a proto ji lze využít ke čtení i zápisu. Třída `RandomAccessFile` se podobá třídám `FileInputStream` a `FileOutputStream` v tom, že lze určit soubor v nativním systému souborů, jenž se po vytvoření otevře. Je-li vytvořen objekt typu `RandomAccessFile`, musí se uvést, zda se bude ze souboru jen číst nebo i zapisovat. Následující kód vytvoří objekt typu `RandomAccessFile`, který bude určen pouze pro čtení.

```
new RandomAccessFile("soubor.txt, "r");
```

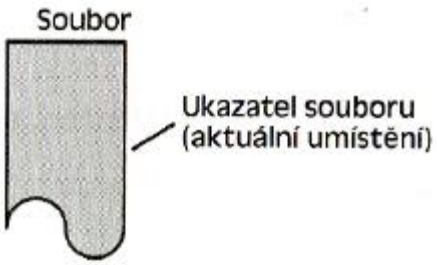
[26] http://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html

Tento příkaz otevře soubor pro čtení i zápis.

```
new RandomAccessFile("soubor.txt, "rw");
```

Po otevření souboru lze využívat vstupně-výstupní operace pomocí běžných metod `read()` nebo `write()`, které jsou definovány v rozhraní `DataInput` a `DataOutput`.

Třída `RandomAccessFile` podporuje koncepci ukazatele souboru (Obrázek 6). Ukazatel souboru určuje aktuální umístění v souboru. Při vytvoření souboru je jeho ukazatel nastaven na hodnotu 0 a označuje začátek souboru. Volání metod `read()` a `write()` posunou hodnotu ukazatele o počet přečtených či zapsaných bajtů.



Obrázek 6 - Soubor s podporou aktuálního ukazatele

Kromě běžných souborových vstupně-výstupních metod, které implicitně nastavují novou polohu ukazatele, obsahuje třída `RandomAccessFile` také tři metody na explicitní manipulaci s ukazatelem souboru.

- `int skipBytes(int)` - přesune ukazatel souboru v předurčený počet bajtů.
- `void seek(long)` - umístí ukazatel souboru těsně před určený bajt.
- `long getFilePointer()` - vrátí aktuální bajtové umístění ukazatele souboru.[1][5]

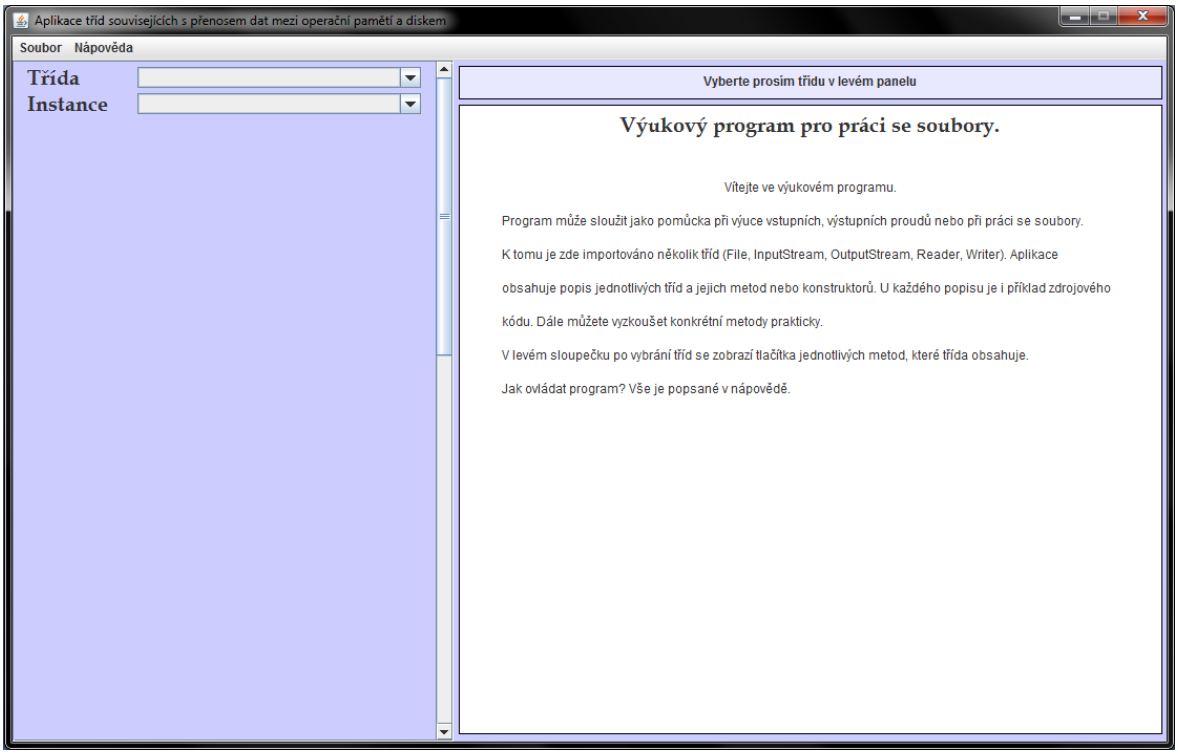
4 Programová část

4.1 Popis a vzhled programu

Zadáním bakalářské práce bylo navrhnout aplikaci zaměřenou na názornou demonstraci konstruktorů a metod tříd jazyku Java určených pro přenos dat mezi operační pamětí a přídavnými zařízeními.

Program je napsán v jazyce Java. Jako vývojové prostředí je použit NetBeans IDE 7.3. Aplikace má navržené grafické uživatelské rozhraní pomocí knihovny Swing.

Vytvořená aplikace slouží jako pomůcka při výuce předmětů používajících jazyk Java. Skládá se z ovládacího panelu, na kterém si uživatel může vybrat příslušnou třídu a její metody. Každá třída, konstruktor a metoda má svůj popis v pravém panelu, kde je implementováno textové pole. Dále obsahuje panel pro zadávání potřebných parametrů jednotlivých metod nebo konstruktorů a stavový panel, který informuje uživatele. V poslední řadě se v aplikaci nachází nápověda pro ovládání programu. Uživatel se v programu dozví, jak využívat jednotlivé třídy, konstruktory tříd a jejich metod. Dále může vyzkoušet jednotlivé třídy v praxi. Na obrázku 7 je vidět vzhled aplikace.



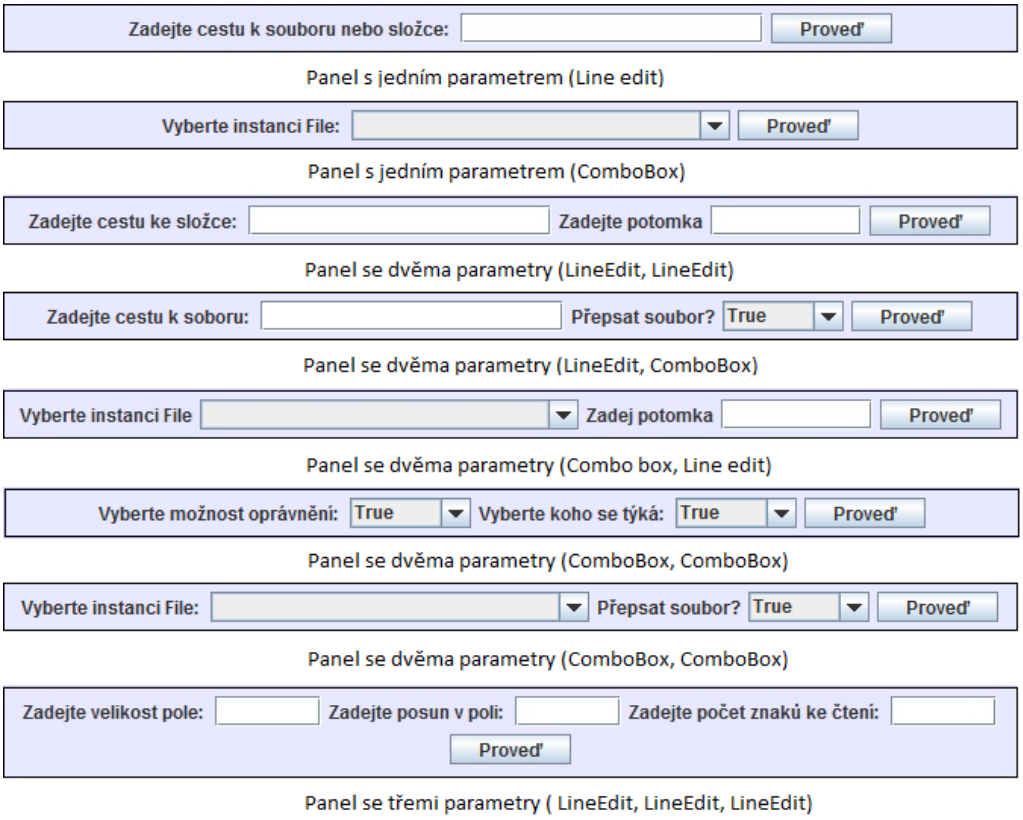
Obrázek 7 - Úvodní stránka aplikace

Celý program je postaven na odchytávání výjimek tak, aby nedocházelo k padání programů díky porušení integrity. Pokud nastane některá z výjimek, uživatel je okamžitě informován pomocí stavového panelu. Například pokud nebyla vytvořena instance příslušné třídy a uživatel chce využívat její metody, bude informován, že nejprve musí vytvořit instanci dané třídy.

4.2 Obsluha programu

Aplikace má většinu ovládání v levém panelu. Zde se nachází ComboBox, který obsahuje názvy tříd. Pokud uživatel vybere jednu z nich, na panelu se vykreslí jednotlivé konstruktory a metody dané třídy. Dále obsahuje ComboBox pro výběr instance. Kdykoliv uživatel vytvoří novou instanci, přidá se právě do tohoto ComboBoxu. Poté uživatel může přepínat mezi jednotlivými instancemi.

Pravá část aplikace obsahuje panel pro zadávání parametrů, stavový panel a textové pole. Pokaždé když uživatel vybere některou třídu, metodu nebo konstruktor, v textovém poli se zobrazí popis vybrané komponenty. Panel pro zadávání parametrů se zobrazí kdykoliv klikneme na některé tlačítko metody nebo konstruktoru, u kterých je zapotřebí zadat vstupní parametr. Panel má několik možností zobrazení (Obrázek 8).



Obrázek 8 - Panel s parametry

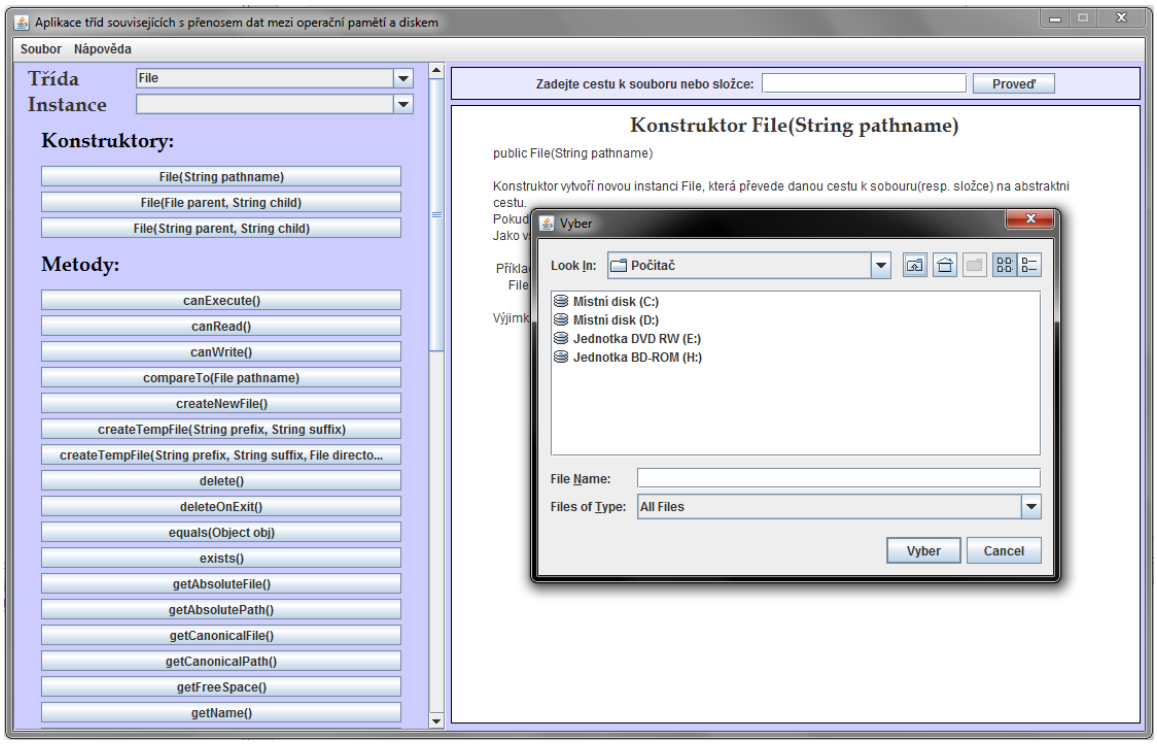
Stavový panel slouží k informování uživatele. Nabývá několika stavů, které jsou barevně rozlišeny (Obrázek 9).

- Zelená - požadovaná operace proběhla v pořádku.
- Červená - operace se nezdařila. Důvod je uveden uvnitř stavového panelu.
- Oranžová - pokud uživatel vybere metodu bez vytvoření instance.

- Žlutá - tento stav se zobrazí jen v některých případech. Například pokud uživatel použije metodu `delete()`. Zobrazí se modální okno s výběrem, zda opravdu chce smazat vybraný soubor nebo složku. Pokud vybere možnost cancel, zobrazí se tento barevný mód.

Obrázek 9 - Stavové panely

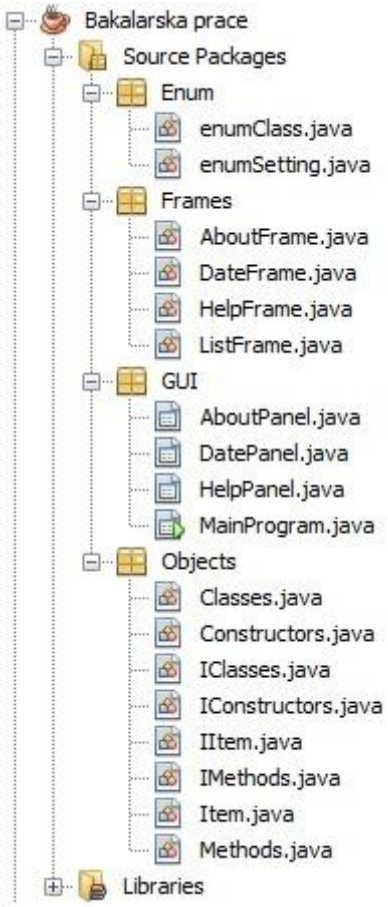
Pokud uživatel potřebuje zadat cestu k některému souboru nebo složce, stačí kliknout na textové pole, které se nachází na panelu parametrů. Poté se zobrazí file dialog pro pohodlné hledání daného souboru či složky (Obrázek 10).



Obrázek 10 - FileDialog

4.3 Struktura programu

Na obrázku 11 je zobrazena struktura jednotlivých balíčků a jejich tříd.



Obrázek 11 - Struktura programu

4.3.1 Balíček Enum

Balíček obsahuje dvě třídy, které jsou typu `enum`.

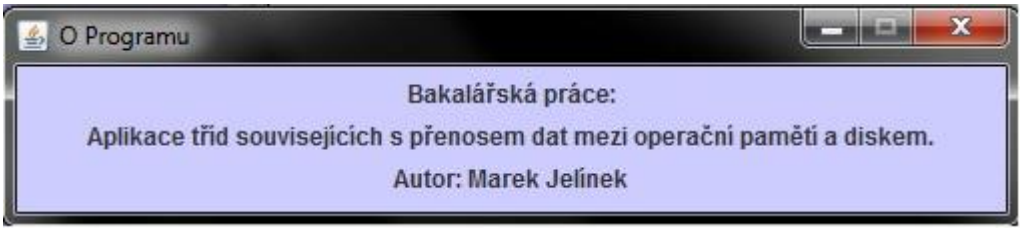
Třída `enumClass` obsahuje názvy implementovaných tříd.

Třída `enumSetting` obsahuje názvy pro nastavení panelu, ve kterém se zadávají vstupní parametry metod nebo konstruktorů.

4.3.2 Balíček Frames

Tento balíček obsahuje třídy, které vytvářejí dodatečné panely, které se zobrazí jen při nějakém úkonu.

Třída `AboutFrame` slouží k vyvolání nového panelu s informacemi o bakalářské práci a jejím autorovi (Obrázek 12).



Obrázek 12 - O programu

Třída `DateFrame` vytváří panel, který slouží k zadání data (Obrázek 13).

Obrázek 13 - Panel pro zadání data

Třída `HelpFrame` vytváří panel, který slouží jako nápověda k obsluze aplikace (Obrázek 14).

Obrázek 14 - Nápověda

Třída `ListFrame` vytváří panel s textovým polem, ve kterém je zobrazen například přečtený text ze souboru (Obrázek 15).

Obrázek 15 - Textové pole pro výpis

4.3.3 Balíček GUI

Balíček GUI se stará o vizuální podobu celé aplikace a také obsahuje třídu `MainProgram`, která zajišťuje chod programu.

4.3.4 Balíček Objects

Tento balíček obsahuje třídy, které uchovávají informace o jednotlivých třídách, konstruktorech a metodách. Hlavní část aplikace se odkazuje na jednotlivá rozhraní.

Třída `Classes` implementuje metody rozhraní `IClasses` a slouží k uchovávání názvů a popisů jednotlivých tříd.

Třída `Constructors` implementuje metody rozhraní `IConstuctors` a slouží k uchovávání názvů a popisů jednotlivých konstruktorů.

Třída `Item` implementuje metody rozhraní `IItem`. Slouží k ukládání jednotlivých tříd a jejich metod a konstruktorů do array listu.

Třída `Methods` implementuje metody rozhraní `IMethods` a slouží k uchovávání názvů a popisů jednotlivých metod

Závěr

Cílem bakalářské práce byl návrh aplikace, která slouží pro podporu výuky v jazyce Java. Zabývá se přenosem dat, kde uživatel může prakticky odzkoušet implementované třídy nebo nastudovat funkčnost z popisu jednotlivých metod, konstruktorů a tříd. Pro usnadnění je vytvořena nápověda, která stručně popisuje ovládání programu.

První kapitola se zabývá proudy. Bylo zde představeno několik streamů a jejich úloha. Dále lehký úvod do serializace objektů a ošetření chyb. Nakonec byla kapitola ukončena popisem jednotlivých tříd vstupních a výstupních proudů. Jako ukázka byl do práce přidán zdrojový kód pro ukládání a čtení ze souboru.

Další kapitola se zabývá skenováním a formátováním textu. Byly popsány metody pro formátování a na ukázku vložen zdrojový kód.

Předposlední kapitola byla věnována práci se soubory. Popisuje třídu File a její metody, které jsou rozděleny podle úkonů. Vytvoření instance pomocí jednotlivých konstruktorů a specifikace abstraktní cesty. Dále popisuje soubory s náhodným přístupem.

V poslední kapitole je popsán vzhled a samotná aplikace. Dále se zabývá obsluhou programu a jako poslední je zde uveden popis jednotlivých balíčků a tříd navržených v programovacím prostředí NetBeans IDE. Aplikace má implementovány čtyři hlavní třídy pro přenos dat a jednu pro práci se soubory. Jsou to třídy `FileInputStream`, `FileOutputStream`, `Reader`, `Writer` a `File`.

Cíle práce byly splněny ve všech bodech.

Program by mohl být v budoucnu rozšířen o další podtřídy s jejich konstruktory a metodami.

Literatura

1. ZAKHOUR, Sharon. *Java 6: výukový kurz*. Vyd. 1. Překlad Jakub Mikulaštík. Brno: Computer Press, 2007, 534 s. ISBN 978-802-5115-756.

2. Linuxsoft, Jelínek. *I/O operace I.* [Online]. 1.6.2005 [cit. 2013-07-30]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=836

3. Java vstupy a výstupy. [Online] [cit. 2013-07-30]. Dostupné z: http://java.vse.cz/pdf/java-vstupy_vystupy.pdf.

4. Java pro začátečníky - Proudy, serializace. [online]. [cit. 2013-07-31]. Dostupné z: http://www.algoritmy.net/article/30183/Proudy-serializace-18

5. Linuxsoft, Jelínek. *Java-práce se soubory*. [Online]. 1.6.2005 [cit. 2013-08-04]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=866

6. Linuxsoft, Jelínek. *I/O operace II.* [Online]. 23.6.2005 [cit. 2013-08-05]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=855

Příloha A – Přiložené CD

CD obsahuje zdrojové kódy aplikace, textovou část ve formátu pdf a spustitelný jar soubor.