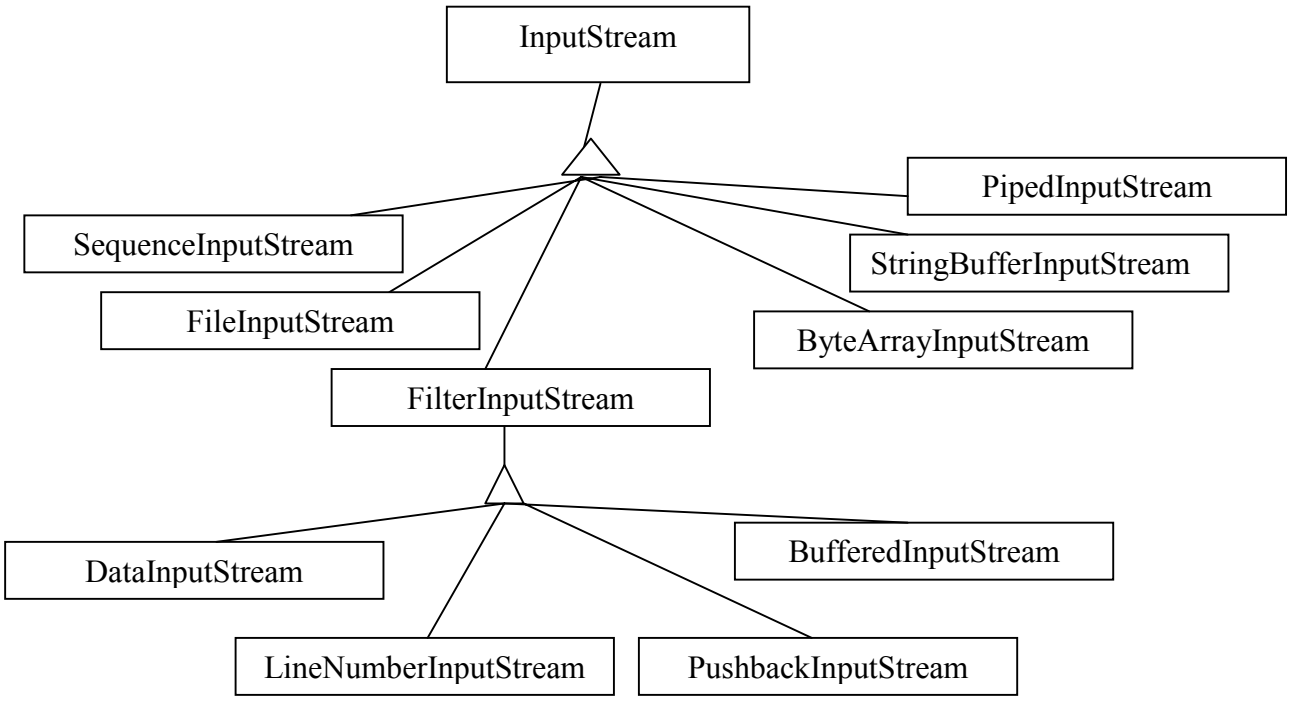
6. Vstupy a výstupy

Pro práci se vstupy a výstupy nám Java poskytuje celou řadu tříd a jejich metod. Jsou uloženy v balíku java.io. Tato knihovna je založena na mechanizmu tzv. vstupních a výstupních proudů (streamů). Java poskytuje v podstatě čtyři základní třídy pro tvorbu proudů. Odlišuje vstupní a výstupní proudy a čtení po bytech nebo znacích. Pro vstupní bytově orientovaný proud je základní abstraktní třídou InputStream a pro bytově orientovaný výstup OutputStream. Pro znakové vstupy pak třída Reader a znakové výstupy třída Writer. Od každé z těchto tříd existuje několik potomků pro čtení/zápis z různých typů vstupních zařízení (soubory, roury, ...). Každá z abstraktních tříd (InputStream, OutputStream, Reader a Writer) má specifického potomka – abstraktní třídu, která je předkem filtrů tj. tříd, které přidávají další funkce pro jednotlivé potomky příslušné třídy. Mimo třídy pro proudy jsou v balíku java.io také třídy File (pro práci se soubory a adresáři), třída StreamTokenizer (pro rozdělení vstupního proudu na části), třídy definující různé výjimky a některé další.

Vstupní proudy

Pro vstupy tedy slouží proudy založené na třídě InputStream nebo Reader. Na obrázku vidíme třídu InputStream a její potomky. Struktura dědičnosti je u třídy Reader obdobná.



obrázek 3 Hierarchie 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.

Třída	použití
ImputStream	abstraktní třída, která definuje základní metody pro čtení po bytech
FileInputStream	čtení ze souboru, parametrem konstruktoru je String se jménem souboru nebo object typu File
PipedInputStream	čtení z roury (z objektu, do kterého zapisuje PipedOutputStream)
SequenceInputStream	vytvoří jeden vstupní proud ze dvou vstupních proudů, které jsou parametrem konstruktoru
ByteArrayInputStream	čtení z pole bytů v paměti, které je parametrem konstruktoru
StringBufferInputStream	převede String na InputStream, doporučuje se používat místo toho StringReader

tabulka 8 Třídy pro vstup 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 bytů zpět do vstupního proudu

tabulka 9 Třídy přidávající funkčnost pro čtení po bytech

Pro čtení po znacích byla vytvořena třída Reader a její potomci. Měly by se používat vždy, kdy se čte text, neboť v této třídě je garantována správná obsluha znakových sad a převod textu do vnitřního kódování Javy (do znakové sady Unicode). V následující tabulce je přehled tříd pro vytvoření readeru a jejich srovnání s potomky třídy InputStream:

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 konstruktoru je String se jménem souboru nebo object typu File	FileInputStream
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 10 Třídy pro čtení po znacích a jejich obdoba pro čtení po bytech

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

tabulka 11 Třídy pro rozšíření funkčnosti readeru

Ke třídám SequenceInputStream a DataInputStream neexistují odpovídající readery.

Čtení ze souboru

Nejčastěji používaným vstupem je textový soubor. Pokud chceme přečíst tento soubor po řádcích, je nutné si pro to vytvořit vhodný vstupní proud.

Pro čtení po znacích se souboru A.TXT si můžeme připravit instanci třídy **FileReader** takto:

```
FileReader ctiZnak = new FileReader ("a.txt");
```

a poté lze číst jednotlivé znaky z tohoto souboru pomocí metody **read ( )**

```
int znak = ctiZnak.read ( );
```

Většinou však chceme číst po řádcích a pro takovéto čtení musíme instanci třídy FileReader ještě "zabalit" do filtru **BufferedReader**, který nám to umožní:

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

a pak použít metodu **readLine ( )**

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

Obě tyto třídy poskytují také metodu close() pro zavření souboru. Pro testování konce souboru používáme při čtení po bytech hodnotu **–1** nebo hodnotu **null** při čtení po znacích. Následující kód ukazuje jak přečíst a na konzoli vypsat textový soubor a.txt.

```
import java.io.*;
public class Cteni {
     public static void main (String [] args){
        try {
          BufferedReader vstup = new BufferedReader
                                  (new FileReader ("a.txt"));
          String s;
          while ((s = vstup.readLine()) != null)
            System.out.println (s);
          vstup.close();
        }
        catch (IOException e){
          System.out.println ("Chyba na vstupu souboru a.txt");
        }
     }
}
```

Jak již bylo řečeno v předchozí kapitole, některé výjimky je nutno ošetřit. K těmto výjimkám patří **IOException** a proto je její odchycení (případně poslání výše) nutné. Bez ošetření výjimky IOException není tento program přeložitelný.

Čtení z konzole

Pro čtení z konzole lze použít **System.in.read(byte b)**. Jak sami vidíte, pro standardní vstup je použita instance in třídy InputStream, tudíž je možno číst pouze po bytech. Tento standardní vstup, stejně jako výstup, otevírá JVM vždy při spuštění programu. Pro přečtení řádky z konzole je tedy nutné tento stream "zabalit" do filtrů. Nejprve je vhodné převést vstup ze čtení po bytech na čtení po znacích pomocí instance třídy **InputStreamReader** takto:

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

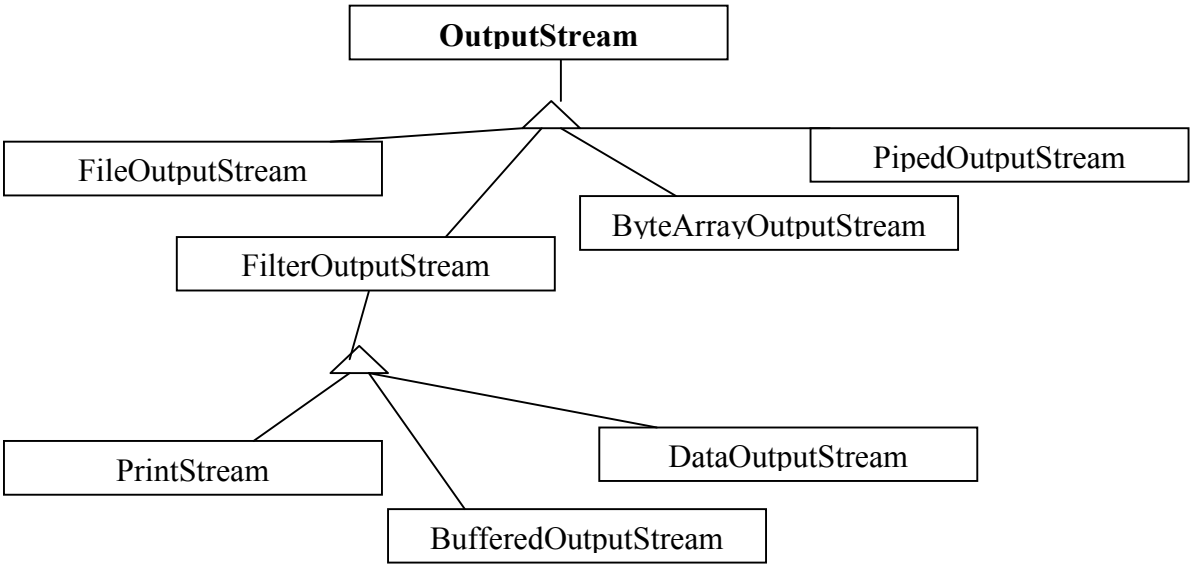
Pro čtení po řádcích pak tento vstup zabalíme ještě do **BufferedReader**u jako při čtení ze souboru. Standardní vstup se zavírá automaticky při skončení programu, není tedy nutné použít metodu close(). Stejně jako při čtení ze souboru musíme ošetřit výjimky **IOException**.

Následující příklad přečte jednu řádku z konzole:

```
import java.io.*;
public class Cteni2 {

      public static void main (String [] args) {
         System.out.print("Zadej text:  ");
         try {
           BufferedReader cti =new BufferedReader
                           (new InputStreamReader(System.in));
           String s = cti.readLine();
           System.out.println (s); }
         catch (IOException e) {
           System.out.println("chyba vstupu");
         }
      }
}
```

Výstupní proudy



obrázek 4 Struktura dědičnosti třídy OutputStream

Obdobně jako u vstupu lze třídy pro výstup rozdělit do čtyř skupin: třídy pro vytvoření výstupního proudu, třídy pro rozšíření funkčnosti výstupního proudu, třídy pro vytvoření Writeru a třídy pro rozšíření funkčnosti Writeru.

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

tabulka 12 Třídy pro zápis po bytech

Třída	použití
FilterOutputStream	předek tříd, které přidávají k OutputStreamu další 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

tabulka 13 Třídy pro přídání funkčnosti pří zápisu po bytech

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 object typu File	FileOutputStream
PipedWriter	zápis do roury (do objektu, ze kterého čte PipedReader)	PipedOutputStream
StringWriter	zápis do bufferu, který může být převeden do objektu String či StringBuffer	-
CharArrayWriter	zápis do pole znaků v paměti	ByteArrayOutputStream

tabulka 14 Třídy pro zápis po znacích

Třída	použití	odpovídající OutputStream
FilterWrite	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

tabulka 15 Třídy pro rozšíření funkčnosti při zápisu po znacích

Třída DataOutputStream nemá odpovídající Writer.

Zápis do textového souboru

Pro zápis do textového souboru použijeme zápis po znacích. Nejdříve musíme soubor pro zápis otevřít, tj. vytvořit výstupní writer. Pokud soubor na disku neexistuje, bude vytvořen nový, pokud existuje, bude přepsán. Jestliže chceme do již existujícího souboru připisovat na konec, musíme použít konstruktor třídy **FileWriter** se dvěma parametry. První parametr je soubor, do kterého budeme zapisovat. Druhý je logická hodnota, která určuje, zda budeme zapisovat za konec souboru nebo původní soubor přepisovat. Hodnota true znamená dopisování na konec souboru.
Pro zápis po znacích do souboru A.TXT si můžeme připravit instanci třídy FileWriter takto:

```
FileWriter pisZnak = new FileWriter ("a.txt");
```

Takto připravený výstup nám umožní zápis po znacích. Výhodnější je však zapisovat po řádcích, proto musíme ještě použít filtr **PrintWriter**, který má metodu **println()** pro zápis celého řádku.

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

Po skončení zápisu nesmíme zapomenout na metodu **close()** pro zavření souboru.
Jako u každé práce s vstupy a výstupy je nutné ošetřit výjimky.
Následující příklad ukazuje zapsání deseti řádek do textového souboru na disk po řádcích (metodou println()).

```
import java.io.*;
public class Soub {

      public static void main(String [] args) {

         try {
           PrintWriter vystup = new PrintWriter
                                (new FileWriter("a.txt"));
           for (int i=1; i<11 ; i++)
             vystup.println("radek "+i);
           vystup.close();
         }
         catch (IOException e) {
           System.out.println("Chyba pri zapisu");
         }
      }
}
```

Výstup na konzoli (System.out) nemusíme nijak "zabalit" protože autoři Javy použili ve třídě System pro proměnnou **out** typ PrintOutputStream, tj. třídu která umí zapisovat celé řádky.

Třída StringTokenizer

V balíku java.util lze najít třídu StringTokenizer, které je určena pro rozdělování textových řetězců na jednotlivé části. Pokud máme např. textový soubor s adresami, kdy na každém řádku jsou údaje o jedné osobě, potřebujeme při načítání souboru oddělit jednotlivé části jako je jméno, příjmení atd. Právě k tomuto účelu slouží třída StringTokenizer a její metody.

V konstruktoru určíme/ řetězec, který se má rozdělit na tzv. **tokeny**. Je možno zadat i rozdělovací znak. Pokud použijete konstruktor bez určení rozdělovacího znaku, jsou standardně použity znaky "\t\n\r", což je tabulátor, mezera a nový řádek.

Třída StringTokenizer poskytuje metodu **hasMoreTokens()** pro určení, zda ještě následuje další část řetězce (vrací hodnotu true nebo false) a metodu **nextToken()**, která vrací hodnotu tokenu jako String.

V následujícím příkladě je ukázáno použití této třídy pro rozdělení řádky textového souboru na části.

Soubor Zvirata.txt má následující obsah:

```
pes Rek
kocka Micka
kocka Mourek
pes Alik
morce Smudla
morce Fousek
kocka Packa
pes Bety
pes Asta
kocka Paty
pes Fik
```

V následujícím programu vypíšeme pouze jména zvířat a bez označení druhu.

```
import java.io.*;
import java.util.*;
public class Tokenizace {

      public static void main(String [] args) {
         try {
           BufferedReader vstup = new BufferedReader
                                  (new FileReader("zvirata.txt"));
           String s,s1,s2 = " ";
           while ((s = vstup.readLine()) != null) {
             StringTokenizer t = new StringTokenizer(s);
             s1 = t.nextToken();
             s2 = t.nextToken();
             System.out.println(s2);
           }
           vstup.close();
         }
         catch (IOException e) {
           System.out.println("Chyba pri cteni/zapisu");
         }
      }
}
```

Výstup programu je následující:

```
Rek
Micka
Mourek
Alik
Smudla
Fousek
Packa
Bety
Asta
Paty
Fik
```

Další třídy a metody pro práci s proudy

Práce se vstupními a výstupními proudy není pouze v balíku java.io, ale i v několika dalších. Existují další metody a třídy pro vytváření vstupních proudů (např. pro čtení ze sítě, pro čtení BLOB z databází) i další třídy pro přidání funkčnosti k proudům (např. komprimace či šifrování).

Čtení ze sítě

Nejdříve si zde uvedeme, jak číst soubor ze sítě. Základem je **třída URL**, v rámci které se zadává adresa souboru, ke kterému chceme přistupovat. Nejjednodušší je zadat textovou adresu jako parametr konstruktoru:

```
URL mojeURL = new URL("http://www.vse.cz/");
```

Pokud zadáme špatný parametr, vyvolá konstruktor výjimku **MalformedURLException**.

Instance třídy URL může vytvořit vstupní proud následujícím způsobem:

```
      InputStream is = mojeURL.openStream();
```

S takto vytvořeným proudem můžeme pracovat jako s kterýmkoliv jiným proudem. Následující příklad vypíše soubor na URL http://www.vse.cz/ na standardní výstup (porovnejte ho s prvním příkladem v kapitole o vstupních souborech):

```
import java.io.*;
import java.net.*;
public class CteniURL {
      public static void main (String [] args){
         try {
           URL mojeURL = new URL("http://www.vse.cz/”);
           InputStream is = mojeURL.openStream();
           BufferedReader vstup = new BufferedReader
                           (new InputStreamReader (is));
           String s;
           while ((s = vstup.readLine()) != null)
             System.out.println (s);
           vstup.close();
         }
         catch (MalformedURLException e) {
           System.out.println ("Chybne URL");
         }
         catch (IOException e){
           System.out.println ("Chyba na vstupu");
         }
      }
}
```

Komprimace a dekomprimace souborů

Třídy a metody pro komprimaci a dekomprimaci souborů typu ZIP a GZIP jsou v balíku **java.util.zip**. Následující příklad zkomprimuje vstupní soubor, jehož jméno je zadáno jako parametr na příkazové řádce do výstupního zkomprimovaného souboru test.gz.

```
import java.io.*;
import java.util.zip.*;
public class GZIPCompress {
      public static void main (String [] args){
         try {
           BufferedReader vstup =
             new BufferedReader (
               new FileReader(args[0]));
           BufferedOutputStream vystup =
             new BufferedOutputStream (
               new GZIPOutputStream (
                 new FileOutputStream("test.gz")));
           int c;
           while ((c = vstup.read()) != -1)
             vystup.write(c);
           vstup.close();
           vystup.close();
         }
         catch (IOException e){
```

```
            System.out.println ("Chyba na vstupu/vystupu");
            e.printStackTrace();
         }
      }
}
```

Třída File

Třída File slouží pro manipulaci se soubory a adresáři. Instancí třídy File může být adresář nebo soubor. Při vytváření instance není nutné, aby soubor (případně adresář) fyzicky existoval na disku. Pro oddělování adresářů a souborů v popisu cesty používáme lomítko / nebo dvě zpětná lomítka (\\ - ve Windows). Lze použít i proměnnou File.separator. Vytvoření instance třídy File je možno pomocí tří různých konstruktorů

- `File (String jmeno);`
- `File (String cesta, String jmeno);`
- `File (File adresar, String jmeno);`

příklad použití	význam
File mujSoubor = new File ("a.txt")	instance mujSoubor nastavena na soubor **a.txt** v aktuálním adresáři
File mojeDopisy = new File("C:"+File.separator+ "dopisy")	instance mojeDopisy nastavena na adresář **dopisy** na disku **C**
File mujDopis = new File ("C:/dopisy/dopis1.txt")	instance mujDopis nastavena na soubor **dopis1.txt** v adresáři **dopisy** na disku **C**
File mujDopis = new File ("C:\\dopisy","dopis1.txt")	instance mujDopis nastavena na soubor **dopis1.txt** v adresáři **dopisy** na disku **C**
File dopis = new File (mojeDopisy,"dopis1.txt")	instance mujDopis nastavena na soubor **dopis1.txt** v adresáři **dopisy** na disku **C**, použili jsme instanci mojeDopisy vytvořenou dříve

tabulka 16 Použití konstruktorů třídy File

Třída File obsahuje metody **isFile()** a **isDirectory()**, které zjistí, zda daná instance třídy File je soubor či adresář (musí již fyzicky existovat na disku). Pokud chceme existenci souboru nebo adresáře ověřit použijeme metodu **exists()**. Všechny tyto tři metody vracejí hodnotu typu boolean.

Pro vytvoření adresáře slouží metoda **mkdir()**, pro vytvoření souboru metoda **createNewFile()**

Zjistit velikost existujícího souboru nebo adresářemůžeme pomocí metody **length()**.Soubor nebo adresář je možno smazat metodou **delete()** nebo přejmenovat pomoci metody **renameTo()**.

Pro výpis adresáře slouží metoda **list()**, která vrací pole řetězců se jmény souborů a adresářů.

Následující program vypíše obsah adresáře dokumenty na disku C.

```
import java.io.*;
public class VypisAdresare {
      public static void main (String [] args) {
         File adresar = new File ("C:/dokumenty");
         String [] obsah = adresar.list();
         for (int i = 0;i < obsah.length;i++)
           System.out.println(obsah[i]);
      }
}
```

Nyní program upravíme tak, aby vypsal pouze adresáře obsažené v adresáři dokumenty.

```
import java.io.*;

public class VypisAdresare2 {
     public static void main (String [] args) {
        File adresar = new File ("C:/dokumenty");
        String [] obsah = adresar.list();
        for (int i = 0;i < obsah.length;i++){
          File prvek = new File (adresar,obsah[i]);
          if (prvek.isDirectory())
            System.out.println(obsah[i]);
        }
     }
}
```

Pokud chceme použít **pro výpis masku** (tj. vypsat pouze některé soubory či adresáře na základě nějaké podmínky) použijeme jako parametr metody list instanci třídy, která má implementováno rozhraní **FilenameFilter.** Toto rozhraní má pouze jednu metodu **boolean accept (File dir, String name)**, kde dir je adresář, který obsahuje testovaný soubor a name je jméno tohoto souboru. Metoda vrací logickou hodnotu zda soubor vyhovuje výběrovému kritériu.