Files and streams in Java

J. Pöial

Stream

- Stream has a producer and possible consumer(s): stream can be viewed as outputstream for the writer and inputstream for the reader
- Stream access is sequential
- Java allows to use two levels of description: "physical" stream (file, array, pipe, ...) and "logical" stream (bytes, primitive data types, objects,Unicode text, ...) - special methods

Types of streams

- input streams and output streams
- buffered streams
- byte streams, data streams and text streams
- file streams, array streams, string streams, piped streams, ...

Low level bytestreams

For reading and writing an unstructured sequence of bytes (subclasses of InputStream / OutputStream):

- FileInputStream / FileOutputStream
- ByteArrayInputStream / ByteArrayOutputStream
- PipedInputStream / PipedOutputStream

Unicode data streams

For reading and writing textual data using 16-bit Unicode symbols (subclasses of Reader / Writer):

- InputStreamReader / OutputStreamWriter
- FileReader / FileWriter
- PrintWriter
- CharArrayReader / CharArrayWriter
- StringReader / StringWriter
- PipedReader / PipedWriter

Data streams for other types

User defined interpretation of data

- FilterInputStream / FilterOutputStream

Streams for Java primitive types

- DataInputStream / DataOutputStream

Java object stream (serialization)

- ObjectInputStream / ObjectOutputStream

Buffered streams

To minimize the number of physical read and write operations:

- BufferedInputStream / BufferedOutputStream
- BufferedReader / BufferedWriter

Used as wrappers:

```
BufferedReader inp = new BufferedReader
    (new InputStreamReader(System.in));
```

flush()-method

Files

- File

  exists, canRead, canWrite, isFile, isDirectory, isAbsolute, getName, getPath, getAbsolutePath, getParent, lastModified, length, mkdir, mkdirs, list, renameTo, delete

  File.separatorChar

- RandomAccessFile (implements DataInput, DataOutput: read..., write...)

  getFilePointer, seek, skipBytes, length

More about java.io.*

IOException (<- Exception <- Throwable)

- EOFException
- FileNotFoundException
- InterruptedIOException
  - .....

System.in – InputStream

System.out, System.err – printStream

Print... and ...File classes do not throw exceptions

Reading from System.in

```
System.out.println ("Text input from System.in");
String s;
try {
   BufferedReader inp = new BufferedReader
      (new InputStreamReader(System.in));
   System.out.print ("Type text: ");
   s = inp.readLine(); // we have a line of text now
   System.out.println ("You typed: " + s);
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Output to the text file

```
System.out.println ("Text output to the file");
try {
   PrintWriter outp = new PrintWriter
      (new FileWriter ("text.txt"), true);
   outp.println ("First line");
   outp.println ("Second line and\nThird line");
   outp.close();
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Input from the text file

```
System.out.println ("Text input from the file");
String line;
try {
   BufferedReader input = new BufferedReader
      (new FileReader ("text.txt"));
   while ((line = input.readLine()) != null) {
      // line read, use it
      System.out.println (line);
   }
   input.close();
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Data output to the file

```
System.out.println ("Data output to the file");
try {
   DataOutputStream outpstrm = new DataOutputStream
      (new FileOutputStream ("data.bin"));
   outpstrm.writeInt (1234);
   outpstrm.writeDouble (1.23e4);
   outpstrm.close();
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Data input from the file

```
System.out.println ("Data input from the file");
int n; double d;
try {
   DataInputStream inputstrm = new DataInputStream
      (new FileInputStream ("data.bin"));
   n = inputstrm.readInt();
   d = inputstrm.readDouble();
   inputstrm.close();
   System.out.println ("Result: " + n + " and " + d);
}
catch (EOFException e) { // this is important
   System.out.println ("Unexpected end of file: " + e);
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Data input bytewise

```
System.out.println ("Data input from the file");
int byte1;
try {
   FileInputStream strm = new FileInputStream("data.bin");
   System.out.println ("Bytes are: ");
   while ((byte1 = strm.read()) != -1) {
      // byte1 is here, use it
      System.out.print(Integer.toHexString (byte1) + " ");
   }
   strm.close();
   System.out.println();
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Read into the bytearray

```
System.out.println ("Read the file into the bytearray");
byte[] content;
try {
   FileInputStream p = new FileInputStream ("text.txt");
   content = new byte [p.available()];
   p.read (content);
   p.close();
   // content is here, use it
   System.out.write (content);
}
catch (IOException e) {
   System.out.println ("Input/output error: " + e);
}
```

Read directory content

```
System.out.println ("Reading directory content");
String[] dcontent;
File f = new File (".."); // directory name here
if (!f.exists() || !f.canRead()) {
   System.out.println ("Not readable: " + f);
   return;
}
if (f.isDirectory()) {
   dcontent = f.list(); // content as an array
   for (int i=0; i < dcontent.length; i++)
      System.out.println (dcontent [i]);
} else {
   System.out.println ("Not a directory: " + f);
}
```

Examples

- Using pipes
- Matrix read/write
- Example with filter streams