8 - I/O Extra

# BufferedReader/BufferedWriter:

The Java BufferedReader/BufferedWriter class (java.io.BufferedReader/java.io.BufferedWriter) provides buffering to your Reader/Writer instances. Buffering can speed up IO quite a bit. Rather than read/write one character at a time from/to the network or disk, the BufferedReader/BufferedWriter reads/writes a larger block at a time. This is typically much faster, especially for disk access and larger data amounts.

The Java BufferedReader/BufferedWriter is similar to the BufferedInputStream/BufferedOutputStream but they are not exactly the same. The main difference is that BufferedReader/BufferedWriter reads/writes characters (text), whereas the BufferedInputStream/BufferedOutputStream reads/writes raw bytes.

Except for adding buffering to Reader/Writer instances, a BufferedReade/BufferedWriter behaves pretty much like a Reader/Writer. The BufferedReader has one extra method though, the readLine() method. This method can be handy if you need to read input one line at a time. The BufferedWriter adds one extra method though: The newLine() method which can write a new-line character to the underlying Writer. In addition, you may need to call flush() if you need to be absolutely sure that the characters written until now is flushed out of the buffer and onto the network or disk.

When the BufferedReader/BufferedWriter is closed it will also close the Reader/Writer instance it reads from.

```
BufferedReader bufferedReader =
    new BufferedReader(new FileReader("c:\\data\\input-file.txt"));

String line = bufferedReader.readLine();

BufferedWriter bufferedWriter =
    new BufferedWriter(new FileWriter("c:\\data\\output-file.txt"));
```

# File:

The File class in the Java IO API gives you access to the underlying file system. Using the File class you can:

- Check if a file or directory exists. `boolean fileExists = file.exists();`
- Create a directory if it does not exist. `boolean dirCreated = file.mkdir();` or `boolean dirCreated = file.mkdirs();`
- Read the length of a file. `long length = file.length();`
- Rename or move a file.

`boolean success = file.renameTo(new File("c:\\data\\new-file.txt"));`

- Delete a file. `boolean success = file.delete();`
- Check if path is file or directory. `boolean isDirectory = file.isDirectory();`
- Read list of files in a directory.
  `String[] fileNames = file.list(); File[] files = file.listFiles();`

The File only gives you access to the file and file system meta data. If you need to read or write the content of files, you should do so using either FileInputStream, FileOutputStream or RandomAccessFile.

# FileReader/FileWriter:

The Java FileReader/FileWriter class (java.io.FileReader/java.io.FileWriter) makes it possible to read the contents of a file or write characters to a file as a stream of characters. It works much like the FileInputStream/FileOutputStream except the FileInputStream/FileOutputStream reads/writes bytes, whereas the FileReader/FileWriter reads/writes characters. The FileReader/FileWriter is intended to read/write text, in other words. One character may correspond to one or more bytes depending on the character encoding scheme.

The read() method of the Java FileReader returns an int which contains the char value of the character read. If the read() method returns -1, there is no more data to read in the FileReader, and it can be closed. That is, -1 as int value, not -1 as byte value. There is a difference here!

```
Reader fileReader = new FileReader("c:\\data\\input-text.txt");
int data = fileReader.read();
while(data != -1) {
  //do something with data...
  doSomethingWithData(data);

  data = fileReader.read();
}
fileReader.close();
```

When you create a Java FileWriter you can decide if you want to overwrite any existing file with the same name, or if you want to append to any existing file. You decide that by choosing what FileWriter constructor you use.

```
Writer fileWriter = new FileWriter("c:\\data\\output.txt");
fileWriter.write("data 1");
Writer fileWriter = new FileWriter("c:\\data\\output.txt", true);  //appends to fi
le
Writer fileWriter = new FileWriter("c:\\data\\output.txt", false); //overwrites fi
le
```

The Java FileReader/FileWriter assumes that you want to decode/encode the bytes in the file using the default character encoding for the computer your application is running on. This may not always be what you want, and you cannot change it! If you want to specify a different character decoding scheme, don't use a FileReader/FileWriter. Use an InputStreamReader on a FileInputStream or an OutputStreamWriter on a FileOutputStream instead. The InputStreamReader/OutputStreamWriter lets you specify the character encoding scheme to use when reading the bytes in the underlying file.

# InputStream:

The Java InputStream class represents an ordered stream of bytes. In other words, you can read data from a Java InputStream as an ordered sequence of bytes. This is useful when reading data from a file, or received over the network.

The Java InputStream class is the base class (superclass) of all input streams in the Java IO API. InputStream Subclasses include the FileInputStream, BufferedInputStream and the PushbackInputStream among others.

read methods:

- `int data = inputstream.read();`
- `int read(byte[])`
- `int read(byte[], int offset, int length)` The read(byte[]) and int read(byte[], int offset, int length) methods return an int telling how many bytes were actually read.

The InputStream class has two methods called mark() and reset() which subclasses of InputStream may or may not support. If an InputStream subclass supports the mark() and reset() methods, then that subclass should override the markSupported() to return true. If the markSupported() method returns false then mark() and reset() are not supported.

The mark() sets a mark internally in the InputStream which marks the point in the stream to which data has been read so far. The code using the InputStream can then continue reading data from it. If the code using the InputStream wants to go back to the point in the stream where the mark was set, the code calls reset() on the InputStream. The InputStream then "rewinds" and go back to the mark, and start returning (reading) data from that point again. This will of course result in some data being returned more than once from the InputStream.

# FileInputStream/FileOutputStream:

The Java FileInputStream/FileOutputStream class makes it possible to read the contents of a file or write a file as a stream of bytes. The FileInputStream class is a subclass of Java InputStream and the FileOutputStream class is a subclass of OutputStreamThis.

Constructors: The first constructor takes a String as parameter. The second FileInputStream constructor takes a File object as parameter.

When you create a FileOutputStream pointing to a file that already exists, you can decide if you want to overwrite the existing file, or if you want to append to the existing file.

```
OutputStream output = new FileOutputStream("c:\\data\\output-text.txt");
OutputStream output = new FileOutputStream("c:\\data\\output-text.txt", true); //a
ppends to file
OutputStream output = new FileOutputStream("c:\\data\\output-text.txt", false); //
overwrites file
```

# ObjectInputStream/ObjectOutputStream:

The Java ObjectInputStream class (java.io.ObjectInputStream) enables you to read Java objects from an InputStream instead of just raw bytes. You wrap an InputStream in a ObjectInputStream and then you can read objects from it. Of course the bytes read must represent a valid, serialized Java object. Otherwise reading objects will fail.

The Java ObjectOutputStream class (java.io.ObjectOutputStream) enables you to write Java objects to an OutputStream instead of just raw bytes. You wrap an OutputStream in a ObjectOutputStream and then you can write objects to it.

Normally you will use the ObjectInputStream to read objects written (serialized) by a Java ObjectOutputStream.

```
ObjectOutputStream objectOutputStream =
            new ObjectOutputStream(new FileOutputStream("data/person.bin"));
objectOutputStream.writeObject(personObject);

ObjectInputStream objectInputStream =
    new ObjectInputStream(new FileInputStream("data/person.bin"));

Person personRead = (Person) objectInputStream.readObject();
```

# PrintWriter:

The Java PrintWriter class (java.io.PrintWriter) enables you to write formatted data to an underlying Writer. For instance, writing int, long and other primitive data formatted as text, rather than as their byte values.

The Java PrintWriter is useful if you are generating reports (or similar) where you have to mix text and numbers. The PrintWriter class has all the same methods as the PrintStream except for the methods to write raw bytes. Being a Writer subclass the PrintWriter is intended to write text.

The PrintWriter has a wide selection of contructors that enable you to connect it to a File, an OutputStream, or a Writer.

< Previous | Next >