Java File: Reading and Writing Files in Java

File handling in Java is not too complicated once you understand a few basic ideas. The key things to remember are as follows.   
  
You can read files using these classes:

* **FileReader** for text files in your system's default encoding (for example, files containing Western European characters on a Western European computer).
* **FileInputStream** for binary files and text files that contain 'weird' characters.

**FileReader** (for text files) should usually be wrapped in a **BufferedFileReader**. This saves up data so you can deal with it a line at a time or whatever instead of character by character (which usually isn't much use).   
  
If you want to write files, basically all the same stuff applies, except you'll deal with classes named **FileWriter**with **BufferedFileWriter** for text files, or **FileOutputStream** for binary files.   
  
  
Reading Ordinary Text Files in Java

If you want to read an ordinary text file in your system's default encoding (usually the case most of the time for most people), use **FileReader** and wrap it in a **BufferedReader**.   
  
In the following program, we read a file called "temp.txt" and output the file line by line on the console.

import java.io.\*;

public class Test {

public static void main(String [] args) {

// The name of the file to open.

String fileName = "temp.txt";

// This will reference one line at a time

String line = null;

try {

// FileReader reads text files in the default encoding.

FileReader fileReader =

new FileReader(fileName);

// Always wrap FileReader in BufferedReader.

BufferedReader bufferedReader =

new BufferedReader(fileReader);

while((line = bufferedReader.readLine()) != null) {

System.out.println(line);

}

// Always close files.

bufferedReader.close();

}

catch(FileNotFoundException ex) {

System.out.println(

"Unable to open file '" +

fileName + "'");

}

catch(IOException ex) {

System.out.println(

"Error reading file '"

+ fileName + "'");

// Or we could just do this:

// ex.printStackTrace();

}

}

}

If "temp.txt" contains this: 

I returned from the City about three o'clock on that

May afternoon pretty well disgusted with life.

I had been three months in the Old Country, and was

fed up with it.

If anyone had told me a year ago that I would have

been feeling like that I should have laughed at him;

but there was the fact.

The weather made me liverish,

the talk of the ordinary Englishman made me sick,

I couldn't get enough exercise, and the amusements

of London seemed as flat as soda-water that

has been standing in the sun.

'Richard Hannay,' I kept telling myself, 'you

have got into the wrong ditch, my friend, and

you had better climb out.'

The program outputs this: 

I returned from the City about three o'clock on that

May afternoon pretty well disgusted with life.

I had been three months in the Old Country, and was

fed up with it.

If anyone had told me a year ago that I would have

been feeling like that I should have laughed at him;

but there was the fact.

The weather made me liverish,

the talk of the ordinary Englishman made me sick,

I couldn't get enough exercise, and the amusements

of London seemed as flat as soda-water that

has been standing in the sun.

'Richard Hannay,' I kept telling myself, 'you

have got into the wrong ditch, my friend, and

you had better climb out.'

### Reading Binary Files in Java

If you want to read a binary file, or a text file containing 'weird' characters (ones that your system doesn't deal with by default), you need to use **FileInputStream** instead of FileReader. Instead of wrapping FileInputStream in a buffer, FileInputStream defines a method called **read()** that lets you fill a buffer with data, automatically reading just enough bytes to fill the buffer (or less if there aren't that many bytes left to read).   
  
Here's a complete example. 

import java.io.\*;

public class Test {

public static void main(String [] args) {

// The name of the file to open.

String fileName = "temp.txt";

try {

// Use this for reading the data.

byte[] buffer = new byte[1000];

FileInputStream inputStream =

new FileInputStream(fileName);

// read fills buffer with data and returns

// the number of bytes read (which of course

// may be less than the buffer size, but

// it will never be more).

int total = 0;

int nRead = 0;

while((nRead = inputStream.read(buffer)) != -1) {

// Convert to String so we can display it.

// Of course you wouldn't want to do this with

// a 'real' binary file.

System.out.println(new String(buffer));

total += nRead;

}

// Always close files.

inputStream.close();

System.out.println("Read " + total + " bytes");

}

catch(FileNotFoundException ex) {

System.out.println(

"Unable to open file '" +

fileName + "'");

}

catch(IOException ex) {

System.out.println(

"Error reading file '"

+ fileName + "'");

// Or we could just do this:

// ex.printStackTrace();

}

}

}

I returned from the City about three o'clock on that

May afternoon pretty well disgusted with life.

I had been three months in the Old Country, and was

fed up with it.

If anyone had told me a year ago that I would have

been feeling like that I should have laughed at him;

but there was the fact.

The weather made me liverish,

the talk of the ordinary Englishman made me sick,

I couldn't get enough exercise, and the amusements

of London seemed as flat as soda-water that

has been standing in the sun.

'Richard Hannay,' I kept telling myself, 'you

have got into the wrong ditch, my friend, and

you had better climb out.'

Read 649 bytes

Of course, if you had a 'real' binary file -- an image for instance -- you wouldn't want to convert it to a string and print it on the console as above. 

### Writing Text Files in Java

To write a text file in Java, use **FileWriter** instead of FileReader, and **BufferedOutputWriter** instead of BufferedOutputReader. Simple eh?   
  
Here's an example program that creates a file called 'temp.txt' and writes some lines of text to it. 

import java.io.\*;

public class Test {

public static void main(String [] args) {

// The name of the file to open.

String fileName = "temp.txt";

try {

// Assume default encoding.

FileWriter fileWriter =

new FileWriter(fileName);

// Always wrap FileWriter in BufferedWriter.

BufferedWriter bufferedWriter =

new BufferedWriter(fileWriter);

// Note that write() does not automatically

// append a newline character.

bufferedWriter.write("Hello there,");

bufferedWriter.write(" here is some text.");

bufferedWriter.newLine();

bufferedWriter.write("We are writing");

bufferedWriter.write(" the text to the file.");

// Always close files.

bufferedWriter.close();

}

catch(IOException ex) {

System.out.println(

"Error writing to file '"

+ fileName + "'");

// Or we could just do this:

// ex.printStackTrace();

}

}

}

The output file now looks like this (after running the program): 

Hello there, here is some text.

We are writing the text to the file.

### Writing Binary Files in Java

You can create and write to a binary file in Java using much the same techniques that we used to read binary files, except that we need **FileOutputStream** instead of **FileInputStream**.   
  
In the following example we write out some text as binary data to the file. Usually of course, you'd probably want to write some proprietary file format or something. 

import java.io.\*;

public class Test {

public static void main(String [] args) {

// The name of the file to create.

String fileName = "temp.txt";

try {

// Put some bytes in a buffer so we can

// write them. Usually this would be

// image data or something. Or it might

// be unicode text.

String bytes = "Hello theren";

byte[] buffer = bytes.getBytes();

FileOutputStream outputStream =

new FileOutputStream(fileName);

// write() writes as many bytes from the buffer

// as the length of the buffer. You can also

// use

// write(buffer, offset, length)

// if you want to write a specific number of

// bytes, or only part of the buffer.

outputStream.write(buffer);

// Always close files.

outputStream.close();

System.out.println("Wrote " + buffer.length +

" bytes");

}

catch(IOException ex) {

System.out.println(

"Error writing file '"

+ fileName + "'");

// Or we could just do this:

// ex.printStackTrace();

}

}

}

Wrote 12 bytes

The output file 'temp.txt' now contains: 

Hello there