File I/O

Sources:
http://docs.oracle.com/javase/tutorial/essential/io/fileio.html
Big Java 4th Edition

Summary:

File I/O is based on the concept of taking an input or output file and then doing something based on that file. Java natively supports this through the path and file classes. Modern file structures typically are hierarchal, in that they build and continue off of a root directory. All folders will descend from the root directory, regardless of how many folders or files there are in the entire file system. Files are located through their path based off the root directory, connecting the file's folder and the root folder. Java can then assign the path to a path object which can return a file object using toFile(). Java can also create file objects directly by using Strings representing the path. JFileChooser is built into Swing to allow easy file navigation and access.

Java is capable of manipulating files, such as changing a files contents, copying a file to another distinct folder, or analyzing a files contents, using the File class and streams. Java has a number of ways of accessing these contents. BufferedReader and BufferedWriter are capable of accessing characters in blocks. They read data from or write data to a memory area known as a buffer, and the native input or output API is called only when the buffer is empty or full. Input Scanners are another way of reading from files through direct user input. PrintStream returns a text output stream, and the PrintWriter class prints formatted representations of objects to a text output stream. Random access does allow nonsequential modification of files as well.

As stated above, the File class often takes a string containing the file name as a parameter for its constructor. These file names often contain backslashes, which have a specific meaning in Java when contained within a string. In such instances, the backslash must be doubled.

For example:
"C:\Program Files\Eclipse\MyFile.txt"
Should be written as:
"C:\\Program Files\\Eclipse\\MyFile.txt"

If the filename specified does not exist, a FileNotFound exception will be thrown. When a reader has reached the end of the file, an EOFException will be thrown. IOExceptions can occur when input comes across an unexpected file or it can't access it. Your programs should be prepared to handle such exceptions.

File streams can process input and output in different ways. A FileInputStream class, for example, processes the input one 8-bit byte at a time and holds the character in the last 8 bits. A FileReader processes input one character at a time, holding the character in the last 16 bits. A line stream processes input one line at a time, storing it in String form. This might be more appropriate for a large text file. Other stream classes exist and may be more appropriate depending on the expected input.

Object streams may be used to write an object to a file. These classes must implement the Serializable interface to simplify the data contained by a complex object.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class testPathing {

	public static String pathTest = "";
	public File f;
	static Path p1 = Paths.get("C:\\home\\joe\\foo");

	public testPathing() {

		pathTest = this.p1.toString();
		this.f = this.p1.toFile();
	}
	public static void main(String args[]) {
		BufferedReader inputStream = null;
		 PrintWriter outputStream = null;

		System.out.println(pathTest);
		try{
			 inputStream = new BufferedReader(newFileReader(
			"C:\\Users\\zhouj\\Documents\\CSSE221\\FileIO_2\\src\\TheThreeLittlePigs"));
			outputStream = new PrintWriter(new
			FileWriter("C:\\Users\\zhouj\\Documents\\CSSE221\\FileIO_2\\src\\ThreeLittlePigsOut"));

			String line = null;
			while ((line = inputStream.readLine()) != null) {
				System.out.println(line);
				outputStream.println(line);
		}
		if (inputStream != null) {
			inputStream.close();
		}
		if (outputStream != null) {
			outputStream.close();
		}

		 catch(IOException e){
			System.err.println(e);
		 }
	}
}
```

JFileChooser: