21 Writing Text Files

1 Creating text files: PrintWriter

We have employed text files by redirecting the standard input and output, but this way of handling files has limitations. For example, it prevents the program outputting to both a file and the screen, and it doesn't allow us to create more than one file. We present ways of creating and reading files under program control. For the moment we will be creating *text files,* by which is meant that the files are composed of normal text such that they can be read using a text editor. In fact a text file created by a program is indistinguishable from one created with a text editor. Later we will see alternatives to text files.

Handling text files in Java requires knowledge of two classes: class `PrintWriter` is used for creating and writing to text files, and class `Scanner` is used for reading them. We create a text file by instantiating **`PrintWriter`**:

PrintWriter(*String*) throws IOException

`new PrintWriter(s)` creates a file called `s` on the disk. For example, we create a text file called `data.txt` as follows:

```
PrintWriter myFile = new PrintWriter("data.txt")
```

Note that the creation of a text file may generate an I/O exception which the programmer must be prepared to deal with. When the `PrintWriter` object is created, an empty file is simultaneously created in the same folder (directory) as your program (i.e. in the same directory as the .class file). The name of the file in the above example is `data.txt`, but you can choose any name as long as it is permitted by the operating system (most operating systems have rules, such as forbidding names containing '/' or spaces). The name doesn't have to have a `.txt` suffix or any suffix at all, unless required by the operating system. However, in some systems it is common to include a suffix such as `.txt` to indicate that the file is a text file. If the file is to be created in some other folder, the *path name* of the file should be supplied. A typical path name in Windows is `c:\mybox\myjava\data.txt` – this indicates the file called `data.txt` is located in a folder called `myjava`, and this folder is in turn contained within folder `mybox` located on drive `c`. In Java, it is always acceptable to use forward slashes in place of backslashes in path names, as in `c:/mybox/myjava/data.txt`. Indeed it is more convenient to do so because Java requires that backslashes be duplicated when they are included in strings, as in `"c:\\mybox\\myjava\\data.txt"`. In Unix, a typical path name is `/users/students/smith/mybox/myjava/data.txt`. The following example illustrates the use of a path name:

```
PrintWriter myFile = new PrintWriter("c:/mybox/myjava/data.txt")
```

In informal discussion we often talk about a file on disk by referring to the Java object associated with it. Given the preceding declaration, for example, we may use `myfile` in place of `data.txt` to refer to the file.

Once the file is created, you can write text to it using the following `PrintWriter` methods

void print(*String*)
void println(*String*)
PrintWriter printf(*String,Object...*)

`f.print(s)` and **`f.println(s)`** each append `s` to text file `f`, and in the case of `println()` terminates the line. In Windows, lines are terminated by a carriage return character (`'\r'`) followed by a newline character (`'\n'`), whereas in Unix just a single carriage return character is used. For example, if integer variable `n` contains 3 then the statement

```
myFile.println("Result = " + n);
```

appends `"Result = 3"` to file `myFile` and terminates the line. There are versions of `print()` and `println()` for all the basic types:

void print(*int*)
void println(*int*)
void print(*double*)
void println(*double*)
...

`f.printf()` is used just like `System.out.printf()`, except that the output is directed to the file identified by `f`. For example, `myFile.printf("Result = %d days", n)` appends `"Result = 3 days"` to file `myFile` assuming variable `n` contains 3. The value returned by `printf()` is not of interest and we nearly always ignore it. Its parameters (other than the first) are declared to be of type `Object` which for our purposes just means that any type of argument can be supplied in practice.

When you have finished writing to the file you must invoke the following method in `PrintWriter`:

```
void close() throws IOException
```

`f.close()` *closes* text file `f`. If you do not close the file, data may be lost, so don't forget it! Once you close a file you may not write to it further unless you "open" it again (see below).

Java's classes are organised into *packages*. Classes concerned with files, such as `PrintWriter`, are located in a package called `java.io` and must be made available by *importing* them. You can import all the classes in package `java.io` by writing:

```
import java.io.*;
```

at the start of your program.

Example 1: passes and failures file

As an example, the program below reads a sequence of student records from the keyboard and creates two text files – a *passes* file and a *failures* file. Each student record occurs on a line and consists of forename, surname, and a percentage mark. A typical input is

```
Bill Smith 69
Jill Wright 43
Anne Butler 89
Max Wallace 38
```

The failures file (which we'll call `bad.txt`) is to contain the names of all students whose mark is less than 45, and the passes file (which we'll call `good.txt`) is to contain the names of all students whose mark is at least 45. The program follows. Observe that a single exception handler is provided rather than one for each file operation. This is the preferred style to avoid cluttering the code with try-catch blocks.

```
import java.io.*;
class TextWrite {
    public static void main(String[] args) {
        try {
            final int passMark = 45;
            PrintWriter passes = new PrintWriter("good.txt");
            PrintWriter failures = new PrintWriter("bad.txt");
            while (!Console.EndOfFile()) {
                String forename = Console.readToken();
                String surname = Console.readToken();
                int mark = Console.readInt();
                if (mark<passMark)
                    failures.println(forename + " " + surname);
                else
                    passes.println(forename + " " + surname);
            }
            passes.close(); failures.close();
        }
        catch(IOException e) {
            System.out.println("File handling failure!");
        }
    }
}
```

Files created using PrintWriter are just regular text files whose contents may be examined by opening them in a text editor (such as WordPad or Notepad in Windows).

The names of the files in the program above (bad.txt and good.txt) have been built into the program. If we prefer, we could allow the user to give the files names of his or her choosing, for example by entering them as command-line arguments:

```
java TextWrite pass.txt fail.txt
```

(note there is no > in the command line because we are not engaging in redirecting the standard output here – each of pass.txt and fail.txt is a string argument to method main() of class TextWrite). In that case, the PrintWriter declarations in the programs should be

```
PrintWriter passes = new PrintWriter(args[0]);
PrintWriter failures = new PrintWriter(args[1]);
```

It is still possible to take input from a text file (students.txt, say) by re-directing the standard input:

```
java TextWrite good.txt bad.txt < students.txt
```

Alternatively, the user could be prompted to key in the names of the files during execution:

```
System.out.print("Enter name of passes file: ");
String passFile = Console.readString();
System.out.print("Enter name of failures file: ");
String failFile = Console.readString();
PrintWriter passes = new PrintWriter(passFile);
PrintWriter failures = new PrintWriter(failFile);
```

At the first prompt, the user should enter, say, good.txt, and at the second bad.txt.

Appending to existing text files

You may append text to an already existing text file. You "open" the file (i.e. make it available to the program) by creating a PrintWriter object as before, but this time wrapping the filename in some additional text:

PrintWriter(new FileWriter(*String*, *boolean*)) throws IOException

`new PrintWriter(new FileWriter(s,true))` makes the text file named `s` on the disk available for output, such that all data will be written at the end of the file right from the start. The file will normally already exist, but if not then an empty file with the given name will be created. As an example:

```
PrintWriter myFile = new PrintWriter(new FileWriter("data.txt", true))
```

opens file `data.txt` for appending. Although class `FileWriter` is used as an intermediary in the constructor; no further knowledge of it is needed in order to use class `PrintWriter`.