Advanced Programming

Lab Sheet 13 – I/O

Introduction

This lab is designed to give you some practice manipulating the filesystem, and reading and writing files. When first working with I/O, it is common for students to encounter difficulty in arranging files in their projects such that they can be manipulated by their code. My recommendation is to create a folder in the root of your eclipse project, alongside rather than inside the src folder. I normally use the name "data", but you can use another name if you prefer.

When you run your Java programs in eclipse, your code runs with a so-called "working directory" of the project root. Thus, if you're using a relative path to access a file, the path should be relative from the project root. In effect, this means that you can acquire a path to a file in the data folder with:

```
Path someVariableName = Paths.get("./data/file.txt");
```

The path String above may look a little odd, particularly if you are used to working with Windows paths, which normally use a backslash (\) instead of a slash (/) as a directory separator. Java has built-in support for converting / separators to \ on Windows, but doesn't always convert \ to / on Linux or Mac. Thus, its best to use / in your code. Starting the path with a period (.) is a good way of indicating that the path is relative to the current working directory.

Task 1: Copying Files

Although it is more common to use shell scripts for small tasks involving extensive file system manipulation, there are still occasions where it is useful to copy, move and delete files inside a Java application. Create some text files, in a data directory, containing a few words of content. Save some of the files with the normal .txt file extension, but save others with a different extension. About 3-4 .txt files and 2-3 others should be sufficient. You should also create another empty directory that you are going to use for your program's output.

Write a program that is capable of looping through all of the files in your data directory, and copy all of the .txt files to your output directory. I would recommend using the DirectoryStream class, an instance of which you can obtain with the Files.newDirectoryStream() static method. You will also need to use the Files.copy() method to actually copy each file.

Task 2: Input & Output Streams

As you saw in the first lecture this week, we handle the input and output of data from files in Java with I/O streams. In this exercise, you are going to use both an input and an output stream to perform some basic processing on each of the line's in a file.

Create a new file in your data folder. This file should be a text file, containing several lines of text. Some of the lines of text should start with a + symbol, and some should not.

Write a program that uses an I/O stream to open your text file, checks each line of the file to see if it starts with a + symbol. If a line starts with a +, you should write the line (including the +) to an output file in the data folder. You can adapt some of the example code in the I/O lecture to handle the reading of the data from the stream, and I would recommend starting this exercise by writing the program such that it outputs lines starting with a + to the console, before starting work on the file writing. Remember to close your streams before the program exits!

Extension: Tidying the Code Up

It would be quite easy to end up with some fairly untidy code on this exercise. There's a special Java language feature, introduced in Java 7, that can be used to help alleviate this problem. The try-with-resources block allows you to automatically close a stream when done, whether or not an exception was thrown. We'll be covering them in lectures later in the term. Research the correct syntax for the try-with-resources block, and tidy your code for the exercise using one for the input stream and one for the output stream.

Task 3: Networking

We use I/O streams for networking as well as file I/O, meaning that we can use similar code to that in the previous exercises to download data from a website. In the I/O lecture, you saw an example using the URLConnection interface to print data from a URL to the console. Adapt this example to create a program that can download data from a URL and save it to a file.

Extension: Creating a Command Line Tool

You can use the String array parameter that is passed to your main() method to find out if your program was run with any additional command line parameters (e.g. java download http://www.kriswelsh.com/ or any other URL). Refactor your program so that it supports the downloading of data from any user-specified URL, and use the "Run Configurations" menu option to specify a URL for testing.

Once this extension is complete, your program resembles the popular cURL and wget command line utilities that are commonly used to download data from websites on the command line.