CSCE 155 - Java

Lab 10 - File I/O

Dr. Chris Bourke

Prior to Lab

Before attending this lab:

1. Read and familiarize yourself with this handout.
2. Review the following free textbook resources:
   - *Thinking in Java* 4th Edition, File I/O section, page 647
   - Oracle's Java File I/O:
     `http://docs.oracle.com/javase/tutorial/essential/io/`

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- Understand the differences between binary and plaintext data
- How to read from a file and process information
- How to write to a file to persist information
- Have some exposure to other topics such as XML and sorting

2 Background

The life span of most program is short–measured in seconds or microseconds. For data to be useful, it needs to last beyond the typical program. This is known as data persistence. One mechanism for persisting data is to store it in a file. Files can generally consist of raw binary data or plaintext. In either case, the data needs to be structured in some manner for a program to be able to read and write it.

In Java, file I/O is facilitated through reading from/writing to streams of data. You are likely already very familiar with the standard input stream and the standard output stream, but there are others as well. In order to read/write text files, Java uses a file object along with another object that can interact with input or output streams. In this lab we'll be using the classes `Scanner` for input, and `PrintWriter` for output. Full details about these classes can be found in the standard Java documentation:

- `Scanner` : `http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html`
- `PrintWriter` : `http://docs.oracle.com/javase/6/docs/api/java/io/PrintWriter.html`

The `Scanner` class provides a useful interface to read data in various formats. You can instantiate it to read from a file as well as from the standard input:

```java
Scanner s = new Scanner(new File("inputFile.txt"));
int a = s.nextInt();
double d = s.nextDouble();
String msg = s.next();
if(s.hasNext()) {
  msg = s.next();
}
...
Scanner getInput = new Scanner(System.in);
System.out.println("Enter an integer: ");
getInput.nextInt();
```

The `PrintWriter` class offers functionality to output plain text data to a file. When it is instantiated with a `File` object, output can be performed using a vararg `printf` method. For example:

```java
PrintWriter output = new PrintWriter(new File("outputFileName"));
output.printf("value = %3d, ", 15);
output.printf("Hello!\n");
```

The class provides many other ways to process output; see the API for full details.

3 Activities

Clone the code for this lab from GitHub using the following URL: `https://github.com/cbourke/CSCE155-Java-Lab10`.

3.1 Plaintext versus Binary Data

In general, data files can contain either binary data (a collection of 0s and 1s) or plaintext data (ASCII). Binary data is generally readable only by a computer or program that interprets the 0s and 1s as different types of data (7-bit ASCII characters, 32-bit integers, etc.) while ASCII text is readable by humans, but may need additional formatting and data conversions to be handled by a program. In this first activity you will get some experience in the contrast between these two types of data.

Instructions

In this exercise, you will work with a pre-written program that opens a file containing census data on states from the 2010 census.

1. Open the `stateData.txt` data file and observe its contents (do not edit this file)

2. Examine what the `StateData.java` program is doing; compile and run it
3. The program will create a binary output file, `stateData.dat` in the same `data` directory.
4. Open `stateData.dat` in Eclipse (right-click it and select "Open With" → "Text Editor" and observe its contents
5. Right click the file in Eclipse and select "Properties". This will tell you the type of file it is as well as the size (in bytes).
6. Answer the questions on your worksheet and move on to the next activity.

3.2 File Output

Extensible Markup Language (XML) is a markup language that defines a set of rules for formatting data in a file that is both human readable and can be processed by a machine. Each piece of data is semantically marked-up to indicate what that data represents. This enables data to be more portable and interoperable across different programs and different programming languages. Many tools and frameworks have been developed around its usage. For example, the state population data may be encoded in XML as follows.

```xml
<STATES>
  <STATE>
    <NAME>Nebraska</NAME>
    <POPULATION>1826341</POPULATION>
  </STATE>
  ...
  <STATE>
    <NAME>Ohio</NAME>
    <POPULATION>11536504</POPULATION>
  </STATE>
</STATES>
```

Instructions

Modify the `StateData.java` program by implementing (and calling) the function:

`public void toXMLFile()`

1. The function should open a file for writing, `stateData.xml`
2. It should create an XML file containing marked up data on all 50 states as in the example above.

3. Hint: the `String` class has a `trim()` method that returns a new identical string with leading and trailing whitespace removed.
4. Answer the questions on your worksheet and demonstrate your program to a lab instructor.

3.3 File Input & Data Processing

Reading data from a file is often done in order to process and aggregate it to get additional results. In this activity you will read in data from a file containing win/loss data from the 2011 Major League Baseball season. Specifically, the file `mlb_nl_2011.txt` contains data about each National League team. Each line contains a team name followed by the number of wins and number of losses during the 2011 season. You will open this file and process the information to output a list of teams followed by their win percentage (number of wins divided by the total number of games) from highest to lowest.

Instructions

1. Open the `MajorLeague.java` source files. Much of the program has already been provided for you, including a convenience function to sort the lists of teams and their win percentages as well as a function to output them.
2. Add code to open the data file and read in the team names, wins and losses and populate the `teams[]` and `winPercentages[]` arrays with the appropriate data
3. Call the sort and output functions to sort and display your results
4. Answer the questions on your worksheet and demonstrate your working program to a lab instructor

4 Advanced Activity (Optional)

1. When we sorted the baseball teams and their win percentages, we had to do all the "bookkeeping" ourselves: that is, we had to swap elements in both arrays to make sure that the *i*-th team name matched up with the *i*-th win percentage. A much better way would have been to define a class to hold the team name and win percentage. Redesign the program to use such a class.
2. In general, there are no restrictions on the length of a line in a plaintext file (and in a data file, there is not even a concept of a "line"). For the best readability, however, it is best to keep lines to a limited, consistent length. One common maximum length for monotype font is 72 characters per line (CPL). A plaintext

file, `star_wars.txt` has been provided (the original draft script of the movie Star Wars) that contains some very long lines. Write a program to read in the file, line by line. If the line exceeds the 72 CPL limit, break it up into multiple lines (but do not break up individual words). Output the resulting well-formatted file to a separate file.