Week 3 Lab - Java Revisited & Java Documentation

COSC244

1 Introduction

The last two labs provided an opportunity for students who have never used Unix before to get familiar with the environment. We will now assume that you are up to speed and ready get on with some Java programming. You will be asked to write a number of small programs during this lab. The best way to get better at programming is to write programs; lots of them. Make sure you keep a copy of each program that you write.

During the first two labs a sensible directory structure was created to keep your labs in `(~/244/01, ~/244/02 etc)`. From now on you should create these lab directories yourself.

2 Assessment

This lab is worth 0.5%

The marks are awarded for the written answers to the preparation questions (worth 0.25%) and completing the programming exercises (worth 0.25%). Unlike the previous two labs, you will not be awarded marks for merely 'giving it a decent attempt'.

The preparation questions should be answered **before** coming to the lab. They should be stored electronically in a plain text file with a .txt suffix. The questions are denoted by a bold **Q** followed by a number in this document (for example, **Q4**). Your preparation questions will be checked at the beginning of the lab. You may be asked to explain your answers to some of the questions.

To get checked off on the programming exercises, you need to create working programs. The demonstrator will look at the source file, ask you to run the program, and may ask questions about the code.

For this lab, as well as all subsequent labs, to get full marks, you must get your work checked off during your assigned lab time. We expect you to spend some time ahead of the lab preparing for it. Normally you can expect to spend around 2 hours of prep time. However, this lab is fairly simple and may not need much time if you are quite competent at Java.

Make sure you get your work signed off by a demonstrator before leaving the lab.

3 Purpose

This lab has two purposes. The first purpose is to help get everyone up to speed on Java programming. The second purpose is to get everyone familiar with looking up the details of classes and methods in the Java API documentation. In later labs, we will provide you some classes to assist in developing your programming exercises. You need to be able to read and understand the provided classes and their methods, and how to use them.

4 Preparation

To prepare for this lab, you need access to a web browser and the Internet.

4.1 Input streams and output streams

Many of the Java programs you write will read some data from an `InputStream`, do some processing on it and then send it to an `OutputStream`. The input could come from one of many sources, e.g. a network, a file, a keyboard or some other input device etc. Likewise output could be sent to many destinations, e.g. a network, a file, the screen etc. Let's begin by studying the System IO streams and some of their methods in the Java API documentation. In a web browser, open the course web page: http://www.cs.otago.ac.nz/cosc244

Click on *Resources*, then click on *Java API Documentation*. Using the top left window, scroll down to the *java.lang* package and click on it. Using the bottom left window, scroll down to the *System* class and click on it. In the right window you should see the three IO streams which belong to the System class.

Q1 What is the type of the data field *in* (`System.in`)?

Click on the type of the data field *in* to visit the documentation for that class and scroll down to the method summaries.

Q2 What is the name of the method which is used to read single bytes from `System.in`?

Q3 What type of value does it return?

Q4 What does it return when there is no more data?

Click on the *System* class in bottom left window to go back to the documentation for that class. In the right window click on the type of the "standard output stream"

Q5 What is the type of the standard output stream (`System.out`)?

Q6 What is the name of the method which is used to write single bytes to `System.out`?

Q7 What type of value does it take?

Q8 Write some code which would read bytes from `System.in` and write them to `System.out`. Don't forget to check for the end of the stream. You should wrap your loop in a *try-catch* block. *(You will be using this code for lab 6)*

```
InputStream in = System.in;
OutputStream out = System.out;...
```

4.2 The *BufferedReader* class

While data is transferred as streams of bytes, it is often more convenient to deal with larger chunks of data, so most languages have some way of buffering the data. Let's look at the *BufferedReader* class documentation. You can find it by clicking on the *java.io* package in the top left window and then looking near the top of the classes in the bottom left window.

Q9 What method would you use to read a line of text?

Q10 What is the return type of that method, and What does it return if there is nothing more to read?

4.3 The *Scanner* class

We often want to deal with input at even higher level, e.g. reading in a certain number of words or a specific set of tokens. It's also nice to have the error checking handled cleanly. The *Scanner* class gives us these capabilities. Look at the *Scanner* class documentation which you will find in the *java.util* package.

Q11 Write a statement to create a scanner to read input from the keyboard.

Q12 What method do you use to find out if a *Scanner* has any more lines to read?

Q13 What method do you use to read a line from a *Scanner*? What does it return?

Q14 What method do you use to read the next token from a *Scanner*? What method should you always call before calling this method?

5 Programming Exercises

All of the programs below should read from `System.in` and write to `System.out`.

5.1 Program 1

The first programming exercise is to write a simple program which reads bytes from an *InputStream* and writes each byte to an *OutputStream*.

5.2 Program 2

Copy your first program to another file. Open it in an editor and change the class name to correspond to the new file name. Modify the program to read input one line at a time using a *BufferedReader* and print the line in reverse (you could use a *StringBuilder* to reverse it).

5.3 Program 3

Write a program which reads input one line at a time using a *Scanner* and prints it out converted to upper case.

Before leaving the lab, show your programs to a demonstrator.

5.4 Optional extension exercise

Write a program which does the following:

- Read input one line at a time using a *Scanner*.
  - Read white space separated words from the line using another *Scanner*.
  - Print out each word followed by a single space, converting each word with length four to ****.
- Print out a newline after each line.