COMP90041 - Programming and Software Development
Submission Instructions

Matthew De Bono

August 17, 2015

1 Before we begin

1.1 Using the "default" package

Do not include "package" statements in your code, or your submission will not be able to run. Always use the default package when creating your Java classes. A package statement will appear at the beginning of your *.java* file, as shown in Figure 1. Please remove any package statements from your code before submitting.

```
1  package myPackage;
2
3  public class HelloWorld {
4
5      public static void main(String args[]) {
6          System.out.println("Hello World!");
7      }
8  }
```

Figure 1: Use of a non-default package; please don't do this

1.2 Types of Input

In this subject, we make a distinction between two types of input: Command line arguments, and User input.

1.2.1 Command Line

Lab tasks that require the use of command line arguments will say explicitly that your program should accept command line arguments, or data will come from the command line".

When writing your code to solve these problems, **do not** use a Scanner. All inputs will be provided via the command line, meaning (if you are using an IDE, like Eclipse or NetBeans) you must set the run configurations of your project to provide arguments.

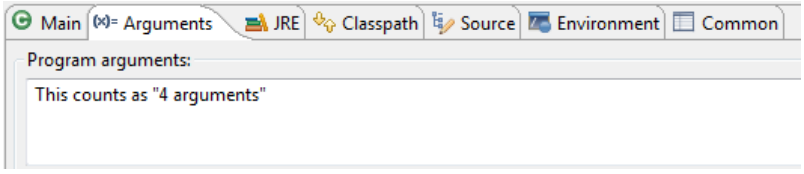

Figure 2: Inputting data using the command line in Eclipse

To do so in Eclipse, click on the "Run" menu, then "Run Configurations". On the following window, click arguments, and you will arrive at the window shown in Figure 2. For NetBeans, click on the "Run" menu, then "Set project Configuration → Customize..." to arrive at the window shown in Figure 3

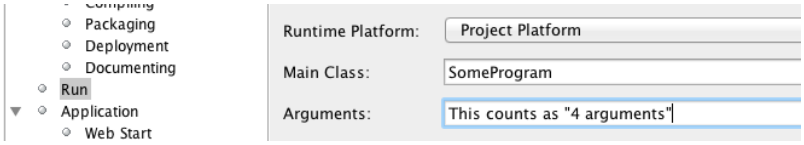

Figure 3: Inputting data using the command line in NetBeans

Each distinct word on the command line is treated as a separate input and will appear as a distinct element in the `args` variable. The exception to this is words within quotation marks; all words within quotes are treated as a single string. For the example input in Figure 3, `args` will look as follows:

args[0] = "This"
args[1] = "counts"
args[2] = "as"
args[3] = "4 arguments"

1.2.2 User Input

Lab tasks that require you to print a prompt or read input **must** be written using a Scanner; there will be no input from the command line. For these tasks, you must use `nextLine()`, `nextInt()`, and other Scanner methods to get input from the "user".

When submitting programs that make use of Scanner for assessed labs, there is obviously no one on the other end typing input for you. When using a Scanner, input may also be provided through a text file. To your code, there is no difference in either method; you must use the Scanner to read input.

2 Submitting from home

If you are submitting using a University lab computer, skip straight to section 3.

Before submitting from home, you **must** log in to one of the University lab computers **at least** once, or you will not be able to access the servers or copy files to them. To submit from home, you first need to download the University's VPN service, which you can get here. Once on the page, click the "student" link, and use your student login to continue to the screen shown in Figure 4. After the site has verified your system, an installer will download to your computer to install the VPN software.

When you run Cisco AnyConnect, you will see the screen shown in Figure 5. Click connect, and enter your student login information to connect to the university servers. If you are successful, you will see the Cisco symbol in your system tray (on Windows). You can now proceed with section 3.

3 Transferring files to server

If you are submitting using a lab computer, make sure your files are stored on the H: drive; if they are anywhere else, you will not be able to find them. In this case, skip to section 4.

In order to submit your assignment, your *.java* files must be on the university servers. The easiest way is to use a file transfer program, such as WinSCP, FileZilla, or MobaXterm. These instructions show WinSCP, but should be almost identical for other programs.

Figure 4: Cisco AnyConnect download screen

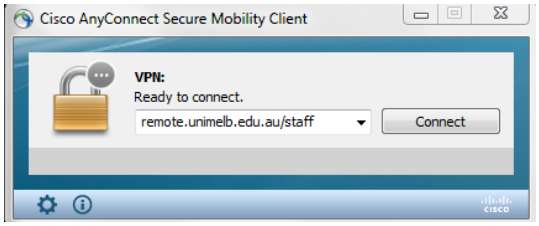

Figure 5: Cisco AnyConnect run screen

When you launch WinSCP you will get the screen shown in Figure 6. In some cases (like in FileZilla) the port number will be empty; make sure to always use port 22. Use your university login to connect to either

`nutmeg.eng.unimelb.edu.au`
`dimefox.eng.unimelb.edu.au`

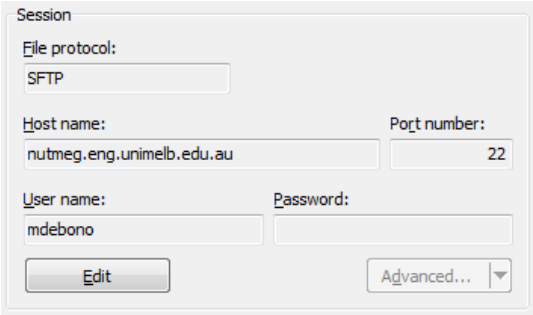

Figure 6: WinSCP login screen

After connecting you will see the screen shown in Figure 7. The left hand explorer window shows files on your local computer, the right hand side are files on the University servers. To transfer your files, simply locate them on the left side and drag them to the right. Your files are now on the server and should appear on the right hand explorer window; you can submit!

4 Submitting your assignment

You must perform this step in order to get marks. Transferring your files to the server is not the same as submitting your files.

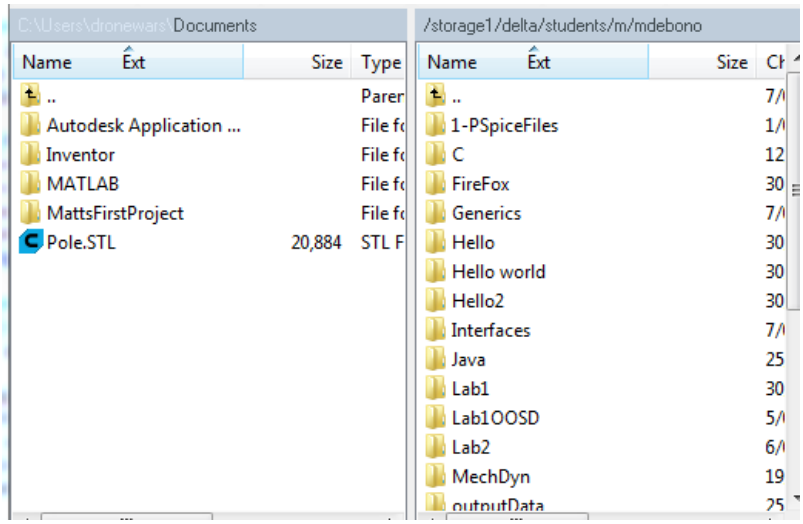

Figure 7: WinSCP file browser

To submit your files, you need to use a terminal program. Mac users may use the inbuilt terminal, but Windows users must download a program capable of SSH connections; these instructions will use Putty.

To connect using the Mac terminal, simply type

`ssh "your username here"@nutmeg.eng.unimelb.edu.au` or
`ssh "your username here"@dimefox.eng.unimelb.edu.au`

(without the quotation marks), and follow the instructions below. When you launch Putty, you will see a similar screen to using WinSCP; use either `nutmeg` or `dimefox` to connect to the servers, and again make sure to use port 22, as in Figure 8. Please note that even though no characters appear on the screen, you are typing your password in.

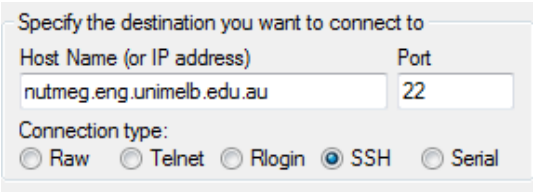

Figure 8: Putty login screen

Once you have connected, you will see the black terminal screen. Before you submit, type **ls** to **list** the files in your current directory. As you can see in Figure 9, there are no *.java* files but there are directories, coloured in blue. If this is the case, use **cd** to **change directory**, and move down into the folder where you put your files. For example, if you put your files in the **Lab1** folder, you would type **cd Lab1/** to move into that folder. Continue using **ls** and **cd** until you have found your files.

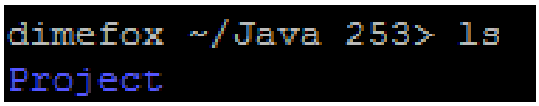

Figure 9: Putty files list; notice there are no *.java* files

Finally, it's submission time! You can use the `submit` command to submit your assignment, which looks like

```
submit mySubject myAssessment myFiles
```

For lab 1, your submit command will be

```
submit COMP90041 lab1 Welcome.java Circle.java
```

Note: COMP90041 is not a folder in your H: drive, it is the subject you are submitting for. If you put your files in a folder named COMP90041, but cannot see them when you type **ls**, then you have not followed the instructions above and won't be able to submit. If you receive an error that says *submit: command not found*, then you need to update your path to be able to find submit; just type in

```
PATH=/usr/local/bin/:$PATH
```

Finally, make sure you submit all required files at the same time; only your last submission is recorded, so if you only submit one file, and try to submit another file in the next submission, this will not work. Submit all files together!

5 Getting feedback

The final step in submitting your assessed labs is getting your feedback, and checking your marks. Each of the assessed labs is marked almost instantly, so don't forget this step! We won't be sympathetic to people who send us emails after the due date, asking why they didn't get full marks ☺

To get your feedback, in Putty or your Mac terminal, type

```
verify COMP90041 lab1
```

Verify will also print out your code, so to make it more readable you can use either of the following. The first will print one screenful of text at a time; you can use *spacebar* to go to the next screen, *b* to go back, and *h* for help. The second version will print everything to "myFeedback.txt", which you can view on a lab computer, or by transferring back to your computer.

```
verify COMP90041 lab1 | less
verify COMP90041 lab1 > myFeedback.txt
```

Finally, when viewing verify, you may see a **diff** message, which shows you where your output has differed from the expected output, but only if there are differences detected. As mentioned above, if you have a package statement at the top of your *.java* file, your code will not execute correctly, and will not give any output.

In the **diff statement**, lines with a minus sign (-) are the correct answer. Lines with a plus (+) sign are lines printed by your code. Compare the plus and minus lines to identify where you may have errors. If you do not have any plus lines, you have not payed attention, and you have a package statement in your *.java* file. The most common **diff** error to receive is "no newline character at end of output". To fix this, make sure that your final print statement is either System.out.println, or you place "\n" at the end of the string.

Congratulations on submitting your assignment! Good luck!