SCC110: Software Development Term 2. Java Programming.

Problem Set 2: Your First Taste of Java

Aims

This week's lab aims to:

- Familiarise yourself with the essential Java build tools.
- Write your first Java program.
- Investigate some of the similarities and differences between C and Java.
- Practice spotting and correcting common mistakes in Java programs.

Developing Java Programs

Log in to the lab computer (or the **SCC Ubuntu** unix image on https://mylab.lancs.ac.uk if you are working from home). These computers already have VS Code and the Java Development Kit installed. Alternatively, install the JDK onto your laptop if you have one. See the lecture notes from this week's lecture for guidance on how to do this.

The basic Java development tools run via a command line interface (just like you have used in C and Unix). There are visual development tools to help you with Java programming that you'll learn later in the course, but it's important that you learn the basics first so we'll be working with the command line this term.

We strongly recommend you keep backups of your work, and we will soon be looking at professional tools to help you do this. In the meantime, we recommend you keep your work backed up on your university H drive. **Do not store your work on the desktop of Lancaster University lab machines** - we can't guarantee your work will still be there the next time you look!

Exercise 1: Hello World!

The only way to start programming in a new language is to write Hello World!
But first, you'll need to find somewhere to store your Java files...

- Open a terminal window and navigate to somewhere on your H drive where you want to store your Java programs. Remember that on unix, you use the "cd" command to do this.
- Create a subdirectory called **java** in there to hold all your Java programs.
- Within your **java** directory, make another subdirectory within this for each of the weekly exercises as you are set them. This will help you keep organized!

To write Java programs you need a text editor. We strongly recommend you continue using **VS Code**. To start VS code from the command line, simply type: **code <filename>** Remember that your filename should end with .java!

- Write a simple Java program to print the words "I will be a good programmer!"

- Save your program into the folder you just created

To compile your program, type: **javac <filename>**

Once your program is compiled, take a look at the files in the current directory. Notice the creation of a **.class** file. This is your Java bytecode. The compiler creates one of these for **each** class you define in your program.

To run your program, type: **java <class name>**

Feast upon the output of your truly magnificent program.

Exercise 2: Familiarise yourself with the core Java syntax compared to C.

Review the Java Syntax Reference slide deck supplied alongside this document on moodle, which highlights the key similarities between C and Java. Ask one of your TAs about anything you don't understand.

Exercise 3: Doing your lines…

Imagine you have been very, very naughty and forgotten to indent your code properly.

- Update your program to write the same text precisely 200 times using a 'for loop'. At the start of each line, show a count of the number of lines written so far.

- You were caught using a 'for loop' and this was considered cheating - update your program to achieve the same goal, but without using a for loop.
  HINT: Rewrite your code using a different loop construct…

Exercise 4: Critically analysing Java code

Now you've familiarised yourself with the Java build environment, this exercise is focussed on reading Java programs, identifying common problems with them and correcting those problems. Good programmers often read their own (or each other's) code to identify mistakes early.

We've provided you with five pre-written programs (all available in the "TestPrograms.zip" file alongside this document on moodle). **All of these programs have at least one mistake that means they don't compile and/or don't behave correctly.** Many of the programs also show very poor code style – correct this before you analyse them.

We've also provided you with a spreadsheet in which to record the mistakes that you find (ErrorDescriptions.xls) on Moodle. Download a copy of this spreadsheet – you should store it along with your Java programs for this week's lab exercise.

For each of the programs provided:

- Improve the code style.

- Without compiling, read the code and record in the spreadsheet any errors you can find just by reading it (this is something called **code inspection**)
- Compile the code (at least try to!).
- Update the spreadsheet to include a description of any new errors you find as a result.
- Write down how you propose to correct these errors in the spreadsheet.
- Correct the errors in the code until you have working program.

Exercise 5: How many days in a month

Finally, write a simple Java program to determine how many days there are in a given month. Remember:

Thirty days hath September,
April, June, and November;
All the rest have thirty-one,
Save February, with twenty-eight days clear,
And twenty-nine each leap year.

- Create a new Java program with an appropriate class name and main method.
- Declare an integer variable called *month* in your main method to represent the month (e.g. 1→Jan, 2 →Feb, etc). Set the initial value of this variable to 1.
- Write java code (as concisely as you can) that prints out the number of days in that month. Use as few lines of code as you can possible (Hint: don't forget the || operator).
- Test your program with all the different possible values of month.