School of Engineering and Built Environment

Dept. of Computer, Communications & Interactive Systems

Software Processes and Practices (M3G605258)

Lab Workbook: Exercises for Weeks 1-11
Session 2014/2015

Prepared by Dr. Richard Foley

Table of Contents

<u>Introduction</u>

This booklet consists of the full set of Lab Exercises for weeks 1-11 of the module Software Processes and Practices. The coursework element of this module is a Practical Lab-based Test which will take place during week 12 of the module. Thus, the purpose of this set of weekly lab exercises is to prepare you for that coursework lab-based test.

As has been/will be stated on several occasions, **<u>this module is not a software development module</u>**. I.e. you will not be given a program specification and be asked to develop a program to provide an application. In that sense the module is not about learning all of the detailed constructs and syntax of a programming language or using it to develop applications.

Instead, this module is about the key software engineering processes and practices which "surround", and to a great extent ensure the quality of, software development. One of the core themes of the module is the practice of Test Driven Development and within that the key practical element is the development of test cases and the automatic execution of tests using an appropriate testing framework. In our case we are using the Java based JUnit, which is one of the family of xUnit automated testing frameworks. Testing is actually the "Cornerstone" of good software development and Test Driven Development is an overall practice which goes a long way to ensuring quality software development.

The lab-based practical test, which forms the coursework element of this module, will take place during week 12 of the module. This test will be 100 minutes in total and during that test you will be required to develop several (comparatively short) test programs using JUnit under the NetBeans Integrated Development Environment (IDE).

Thus, these weekly lab exercises have been provided in order for you to develop sufficient skills in JUnit (and the use of the NetBeans IDE) to enable you to undertake and succeed in the coursework lab-based test. This practical work is complemented by the Tutorial work in weeks 1-4 (which aims to revise, refresh or familiarise you with the necessary underlying Java concepts required), and the accompanying Practical Lecture series (Yellow Lectures Notes with 6 lectures on the practical work of the module).

How to use these lab exercises

As indicated above the main aim of these lab exercises are to enable you to develop skills in the use of JUnit for the construction of (comparatively) simple test programs. An early secondary aim of the exercises, however, is to provide you with a short introduction to the use of NetBeans as an IDE so that you understand how it structures java projects and how JUnit links in with the IDE.

However, all of these exercises must be used to best advantage for that purpose. As a set of exercises these provide an ideal opportunity for you to gain feedback on your performance as you progress through the graded stages of learning about (and using) the various JUnit concepts and constructs for the development of test programs. However, if you do not use them properly then you may lose that potential feedback opportunity!

The first thing to note is that the full set of exercises have been provided to you along with an "expected" weekly schedule. This will allow you to work "flexibly" and independently on these exercises, as is the objective of the University's Strategy for Learning (sometimes also referred to as the University's Learning Teaching and Assessment Strategy). Indeed this will allow students to go at a "faster pace" in the practical work of this module than the "expected norm", which (of course) is one of the aims of encouraging an Independent Learning culture in students. This could be extremely useful for your time management since you may find that the first few weeks of a semester has more "free time" as most taught modules don't really start to "warm up" in terms of practical work and coursework assignments until the latter half of the 12-week teaching period.

To assist you in monitoring your progress, I have included a "progress/completion checklist" for you as Appendix 1 of this booklet. You can use this to record when you attempt each exercise and "tick each off" as you complete it. If you have any difficulty as you attempt to complete one, you can record some information about that in the NOTES pages at the end of this booklet after the complete appendices sections.

Also, as is usual in the practical element of any computing-based module, **there are generally more exercises than can be completed in any weekly one-hour long scheduled lab slot**. Thus, whilst you can progress through the exercises "at your own pace" and at "a time and place of your convenience", **you will be expected to, at least, complete all of the exercises from one week to the next as suggested by the "standard" weekly schedule**.

Remember the scheduled (and timetabled) **weekly lab slot represents your best opportunity for gaining direct feedback** related to your performance and understanding from the member of staff who is taking that class. However, if you do not attempt the work independently between labs, then you are unlikely to get much out of that feedback opportunity. If you spend most of the time in a lab slot just trying to read and "plough your way through" the first of the current weekly exercises then you will not actually have much feedback on your understanding nor have much time to speak to the member of staff about any specific issue or meaningful difficulty you may have with a particular exercise. Thus you are best to spend some time (e.g. a couple of hours at least) before your weekly time timetabled lab session undertaking that week's exercises, so that you come to the lab session ready to ask specific questions relating to your understanding of/attempt at the exercises.

Thus, it is important also not to "get behind" in the weekly schedule. Whilst this workbook enables you to "go at your own pace", if that "pace" is significantly slower than the suggested weekly schedule then you will simply be making it more difficult for yourself in a variety of ways.

Finally, from the week 3 exercises onwards, **I will issue (via GCU Learn) a feedback commentary on each set of exercises**, at the end of the week after each set of exercises should have been completed. This will give solutions and additional teaching points about each set of exercises. You should be sure to study that even if you had successfully completed the exercises at the time (and certainly if you had problems) as these will give further reinforcement of the concepts and material.

General Introduction to the lab exercises

The objective of the practical work of this module is NOT to teach you how to develop Java programs. **This is NOT a software development module**! The objective of this module is to understand the process of developing software and in particular the process of building, testing and deploying software systems, which surrounds and underpins the successful development of software. Whilst this will be done within a java based development environment and thus some familiarity of the basics of the java language will be required, it is NOT a software development module and so you will not be being asked to develop any application software. You will however, be learning sufficient java constructs, which builds upon existing basic concepts (i.e. of the nature of the first year module Introduction to Programming) to enable you to develop tests for java classes which will be specified and supplied.

Specifically, the practical work of this module will concentrate on several specific aspects of the process of building, testing and deploying software:

1. Building/Compiling and executing software from within the NetBeans IDE
2. Understanding the software related components which make up a system build and a software project within a modern IDE
3. Testing (java) components through the construction of simple test “harnesses”
4. Using the JUnit test framework to automate the testing process
5. Understanding the use of CVS within the NetBeans IDE to access a central code repository
6. Understanding and using the core set of JUnit framework facilities to develop test programs for given software.

In the lab work for the first week the aim is to achieve 1 and 2.

Important note concerning scheduling and completing the lab work

As with all of the lab exercises, you are expected to complete them in your own study time. It is important to continue to appreciate that each weekly lab work “handout” gives the set of exercises for a COMPLETE week/set of weeks. It is highly unlikely that you will be able to complete them (and then completely understand them) within the one hour timetabled lab hour. Thus you must schedule your own time in advance of/during the rest of the week to complete and more fully study the exercises.

Texts to (Re)Familiarise yourself with the Java Language

For some students it may have been some time since you last saw Java code. For other (normally Direct Entry) students it is possible that you have had little (or no) exposure to the Java language. As I have previously stated this is not a software development course and in reality the level of basic understanding required is of some of the simplest concepts in a(ny) programming language. The first set of tutorial exercises (weeks 1-4) are aimed at testing and developing your understanding of java programming and these should be used by you for feedback for that purpose as you study them. As you undertake these exercises (and study the solutions posted as the module progresses) it may be useful/advisable to refer to one or two very simple (entry level) java programming textbooks to either re-enforce that concept or develop your understanding of it as this feedback on your performance dictates. Two which are of that level, and copies of which are in the Library, are:

Fundamentals of Programming using Java, by Edward Currie
Thomson Learning, April 2006,

Getting Started in Java, by Owen Bishop
Bernard Babani Publishing, September 2005

My advice for the use of such texts would be to use the book's index (at the back) and look up any programming construct which you don't feel that you understand as you progress through the tutorial (and lab) exercises.

Downloading Netbeans for use on your Home PC

In the University Labs (George Moore Buildings) the NetBeans/Java environment is installed on a Virtual Machine image. However, no doubt you will wish to download Java and Netbeans for your own use. Indeed you would be strongly advised to do so. Both are Open Source Software projects. The simplest method is to download both as a single bundle from the appropriate website. The appropriate link is

http://www.oracle.com/technetwork/java/index.html[1]

There should be a large button labelled **Java Downloads** on that home page (at the bottom of a list labelled **Top Downloads**) and this should take you to a page where you can download a **JDK + NetBeans** bundle, e.g.

JDK 8u5 with NetBeans 8.0)[2]

[1] Oracle took over responsibility for java when they acquired Sun Microsystems a few years ago. The direct link for the bundle page is http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html, which you could use directly in your web browser.

[2] At time of printing of this booklet the latest versions of the JDK **was jdk1.8.0_5** and for NetBeans was version **8.0**. Being an open source product, the development community regularly release new revisions to the main versions (i.e. 1.8.0 and 8) of the products.

After accepting the licence agreement, just select the link for your particular machine/operating system to download the executable file which will install both the java language and the Netbeans IDE on your machine. You should note that there are different bundles for 64-bit and 32-bit Windows operating systems. Thus you should check which type of Windows operating system you have installed (if you are downloading a windows bundle). You can check this via the Control Panel facility of Windows, via:

Control Panel -> System and Security -> System

There are Netbeans/Java bundles for both Linux and Mac. I have not tested these lab exercises with either of these versions. However, I would expect them to work.

Using NetBeans in the University PC Labs

In your timetabled labs, we will be using Netbeans version 8 on a Windows 8 virtual machine. Continuing students should already be familiar with this general arrangement. I.e. you first launch the Virtual Machine (VMware) player and then launch the particular virtual machine image with the software you require. The interface for the VMware in session 14-15 may be upgraded from the previous session. However, you will be given specific guidance in the first timetabled lab about this mechanism, in terms of how to launch the VMware player and which VMware image to use.

All of the labs in this booklet presume that you are using that VMWare image and thus many of the screenshots have a Windows 8 "look and feel" to them. (However, I have left some of the "old" ones in since there isn't actually much difference at all once you start up the NetBeans IDE itself.) Furthermore, whilst you will probably be more familiar with the Windows 7 desktop format, there really isn't much different at all, other than slightly different colours and edges to the windows, and the Start/Menu button is in the same place as the Windows 7 version (i.e. very bottom left), but is a different style, i.e.

It is more this style:

Rather Than:

Obviously, your home use will not be via a Virtual Machine, but will have NetBeans (version 8) installed directly on your desktop. Again you should see no real difference, other than simple "cosmetics" in some of the interface/screenshots.

The VMware image, though, is only available in the Computing Labs on your timetable (e.g. in the George Moore Building (GMB) labs). However, you can also use a version of Netbeans (version 7) which is also more than adequate and is available, through a launch mechanism, called Application Jukebox, which is installed in all University PCs. This might be useful for your additional On-Campus independent study of the lab exercises outwith your formal lab times, if you cannot easily get into a GMB lab which has the VMware image.

Full details of how to use Application Jukebox (via the Application Jukebox Portal which is on the standard PCs) is given in Appendix 2 on page 70. Indeed, you may even feel more comfortable using this mechanism even in your timetabled labs. You may do this if you wish; it is really up to you.

However, you MUST be familiar with the NetBeans 8 on the Windows 8 VMware image, since the coursework assessment (i.e. the Class Test on JUnit) which takes place at the end of the teaching period WILL be undertaken on the Windows 8 VMware version.

Week 1: Executing and "testing" simple java programs from the NetBeans IDE

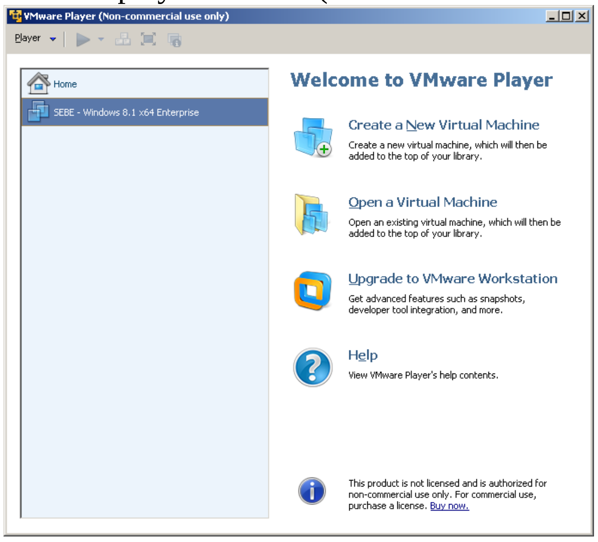
As previously indicated the first week's lab exercises are aimed at enabling you to:
1. Building/Compiling and executing software from within the NetBeans IDE
2. Understanding the software related components which make up a system build and a software project within a modern IDE

Reminder regarding the use of the VMware player to run the java environment

Similar to a number of environments you use in the dedicated Computing Labs, the java/Netbeans environment uses a separate application (VMware Player) which is started first and which looks like a separate instance of Windows running in a window; the java/Netbeans environment is started within this[3].

- The VMware Player is started from[4]:
    - **Start -> All Programs -> VMware** (folder) **-> VMware Player**

This should open the VMware player window (similar to the following)


VMware Player (Non-commercial use only)
Player
Home
SEBE - Windows 8.1 x64 Enterprise
Welcome to VMware Player
Create a New Virtual Machine
Create a new virtual machine, which will then be added to the top of your library.
Open a Virtual Machine
Open an existing virtual machine, which will then be added to the top of your library.
Upgrade to VMware Workstation
Get advanced features such as snapshots, developer tool integration, and more.
Help
View VMware Player's help contents.
This product is not licensed and is authorized for non-commercial use only. For commercial use, purchase a license. Buy now.

There should be a list of several different VMware images in the left hand pane. However, if not, you should use the **Open a Virtual Machine** option and navigate to the **VMware** folder under **Libraries**, open the folder **SEBE-VM-Win8.1x64.2014v1 and then:**

- **Select** and launch (Play) the image **SEBE – Windows 8.1 x64 Enterprise**
- If the License Agreement page appears during any of this process, choose "**Yes, I accept….**" and press **OK**
- When the VMware image loads up, it is often best to set the VMWare application to Full Screen mode, either via the button in its top task bar, or by selecting the appropriate option on the Player pull down menu.

[3] As previously indicated, you can, if you wish, alternatively use an application available throughout the University, called Application Jukebox, to launch a version of NetBeans (7.0) from the standard desktop. All lab exercises will work on this. However, the Coursework will be using the VMware Netbeans 8 version on a Windows 8 virtual machine, and so you do need to be familiar with this. Details of how to use Application Jukebox are at the end of this booklet in Appendix 2.

[4] There may also be a VMware shortcut on the desktop, but launching it from the Start menu is probably the "cleanest" method.

You require to do this at the start of each session when you work on these lab exercises using the University PCs. Remembering that when you logout, all data is lost, you should copy all your work onto a USB drive. The easiest way to do this is to copy the whole folder which contains the complete NetBeans project you have been working on.

For Home use you should download and install the NetBeans/Java bundle directly. The instructions for this were on the previous page (6) of this workbook.

Understanding source to object code translation in a high level language

A source code file contains the text of a high level language. In Java a very simple programme, which will simply printout the message **Hello World** when it executes, is:

```
public class HelloWorldApp
{
      public static void main(String[] args)
      {
            //Display a greeting on the console window
            System.out.println("Hello World");
      }
}
```

You should easily understand this simple program from your initial study of java. However, here, are some reminders (which should all be familiar to you):

A class is the fundamental building block of the java language. A complete program can consist of several (often hundreds of) different classes. These classes are also normally spread across many files and the complete program has to be "linked" together by the build process. This program just consists of a single class which is `public` (i.e. can be usable by all other program elements) called `HelloWorldApp`.

The executable instructions of a class are grouped into separate **methods**. When a method of a class, or class instance (a class instance is often termed an object), is called then its code is executed. Methods can be supplied with parameters (sometimes these are termed arguments) which supply data to be processed by the code of the method, and can return a value which results from the processing of the method.

Within the complete set of classes making up a program there must be one "special" (`static`) method called `main`. It is from this method that the control of the execution of the program will start. The `main` method has a set of standard arguments (which can be supplied as parameters by the command which executes the java object code) and doesn't usually return a value (and thus has the value `void` in its specified return value).

The main method is a static method, which means that it does not "technically" operate on an object (class instance). The main method must be `static` since it has to "exist" as part of the class before any other objects in the execution of the program have been created. Most class/object instances are created dynamically as part of a

developed program's algorithm and how does java (or any Object Oriented) language actually "get the program initially "started". This is a "Chicken and Egg" situation and it is accomplished through the use of this `public static main` method (of which there can only be one instance of within a complete programme, no matter how many files and classes are "linked" together to make up a complete programme/software system. Static methods are only also commonly used in what are termed **utility** classes, such as the `Math` class which would be a class library consisting of a number of basic mathematical functions which can be called and utilised with a program.
The "scope" of both classes and methods (and other program blocks) are identified by a pair of parentheses. In the main method there is a comment (a line commencing with the double backslash – "`//`"). Comments should be used to help communicate and explain the algorithm within the code of programs.

A method will consist of a set of instructions (and data) to be executed. In java, each instruction is terminated by a semicolon. In this case there is only one instruction which is actually a method call itself (in this case the `println` method of the console window, which is represented in java by the `System` object `out`. The `println` method takes a string as a parameter and prints that string, then a newline to the console window.

Enough of the explanations lets create it as a source file.

Ex. 1: Building/Compiling and executing software from within the NetBeans IDE

A source file with the high level language source code is just a standard text file. In java all source files must have the suffix **.java.** In the days before Integrated Development Environments, programmers would use separate editor and compiler utility provided by the host operating system in order to develop and construct software. nowadays, this is integrated into a single development environment. Thus, we are going to now create, (then compile and) then execute this java program using the NetBeans IDE.

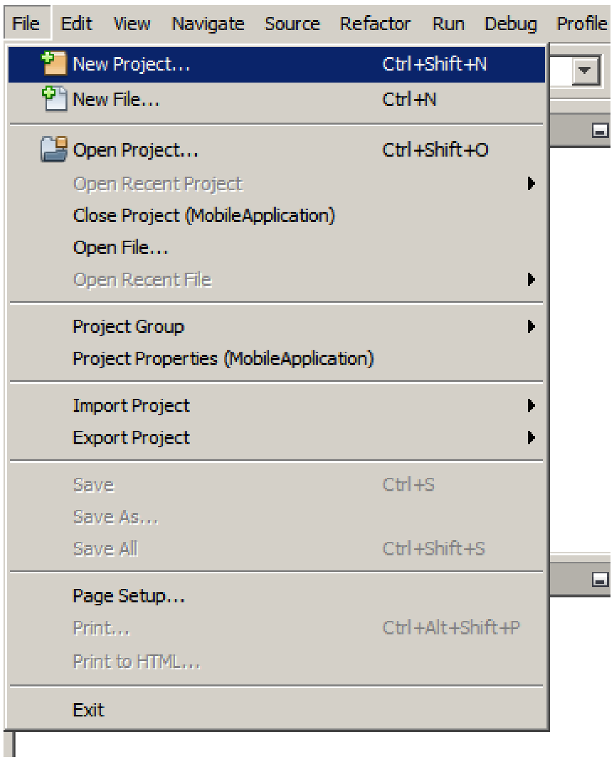
Thus you should now, if you haven't already, launch the Java Applications VMware player (as per the instructions on page 9 of this workbook). Remembering that you might find it useful to make the VMware window full screen.

Being a modern IDE, Netbeans creates an overall Project (folder) within which all of the necessary (sub-folders and) files are contained. These files include, both source and object code for the individual classes which make up the program/application, as well as the distribution file(s) (which contains the linked object code for the software so that it can be used, i.e. imported, by other programs). The project (folder) in Netbeans also contains a variety of additional configuration files, which enables these separate components to be updated and manipulated as required during the development of a program. To do this we will follow the basic "Hello World" tutorial on the Netbeans support pages[5].

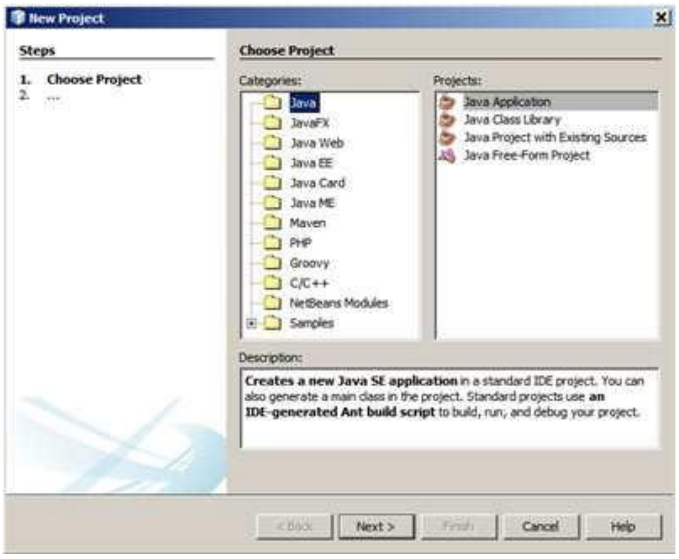
[5] You will find this tutorial at: https://netbeans.org/kb/docs/java/quickstart.html. However, its key points are given here as a set of easy to follow instructions.

To create an IDE project:

- Start NetBeans IDE, via **Start** (button) **-> NetBeans IDE 8.0**
- In the IDE, choose **File -> New Project**, as shown in the figure below.

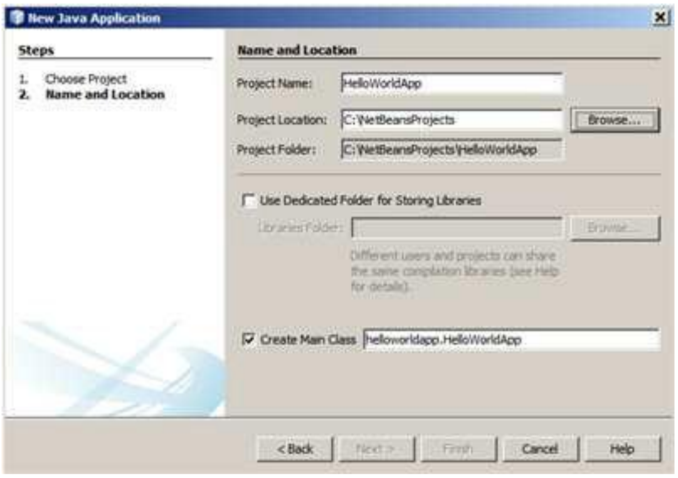


- In the New Project wizard, expand the **Java** category and select (under **Projects:**) **Java Application** as shown in the figure below. Then click **Next**.



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

In the **Name and Location** page of the wizard, do the following (as shown in the figure below):



1. Choose Project
2. **Name and Location**

Name and Location

Project Name: HelloWorldApp

Project Location: C:\NetBeansProjects

Project Folder: C:\NetBeansProjects\HelloWorldApp

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

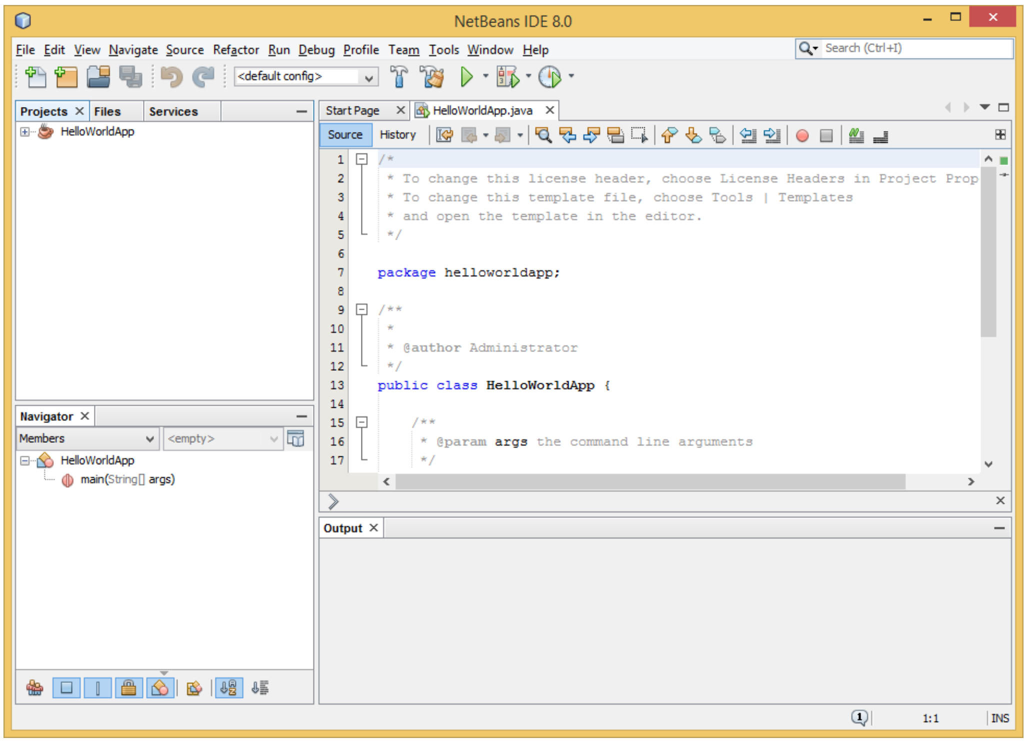
Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class helloworldapp.HelloWorldApp

- In the **Project Name** field, type **HelloWorldApp**.
- Leave the **Use Dedicated Folder for Storing Libraries** checkbox unselected.
- In the **Create Main Class** field, type **helloworldapp.HelloWorldApp** (This entry may auto complete, but just make sure it is as above.)
- Click **Finish**.

The project is created and opened in the IDE. You should see the following components:

- The **Projects** window, which contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.
- The **Source Editor** window with a file called **HelloWorldApp.java** open.
- The **Navigator** window, which you can use to quickly navigate between elements within the selected class.



```
/*
 * To change this license header, choose License Headers in Project Prop
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package helloworldapp;

/**
 *
 * @author Administrator
 */
public class HelloWorldApp {

    /**
     * @param args the command line arguments
     */
```

Because you have left the **Create Main Class** checkbox selected in the **New Project** wizard, the IDE has created a skeleton main class for you. You can add the "Hello World!" message to the skeleton code by replacing, in the ***main*** method the line (you may have to scroll down to see it in the editor window):

// TODO code application logic here

with the lines:

//Display a greeting on the console window
System.out.println("Hello World!");

Then save the change by choosing **File > Save**. (or CTRL + S)
The file should look something like the following code sample.

```
public class HelloWorldApp
{
      public static void main(String[] args)
      {
            //Display a greeting on the console window
            System.out.println("Hello World”);
      }
}
```

You should also note that the java (source code) file has a first line of:

```
package helloworldapp
```

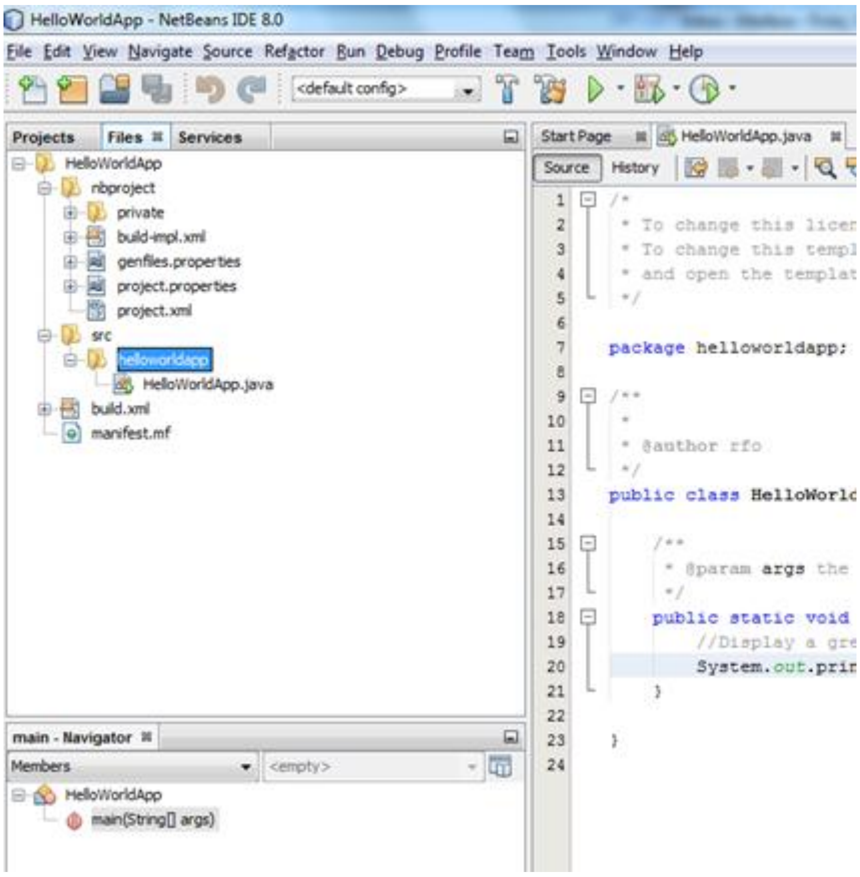
As previously mentioned, in a "real-life" application, there are many classes and thus many files potentially making up the project. In java projects, the source files are "packaged" up in separate folder structures so that they can be more easily managed and distributed between projects and development team members. Thus, in this simple project, the **HelloWorldApp.java** source file has been (automatically) placed in source code folder (called **helloworldapp**). However, before we run this program, let us take a look at the overall folder structure the Netbeans IDE wizard has created for this project. Therefore, do the following

- Select the tab labelled **Files** (found beside the Projects window tab)

This shows the project folder (as a folder called **HelloWorldApp** rather than a java project)

- Now expand (by double-clicking) this **HelloWorldApp** folder structure. Then expand the folders named **nbproject** and **src** (and then the subsequent folder **HelloWorldApp** within the src folder).

You should thus have (something similar to) the following: **(OVERPAGE)**



```
/*
 * To change this licen
 * To change this templ
 * and open the templat
 */

package helloworldapp;

/**
 *
 * @author rfo
 */
public class HelloWorld

    /**
     * @param args the
     */
    public static void
        //Display a gre
        System.out.prin
    }

}
```

The folders you see are as follows:

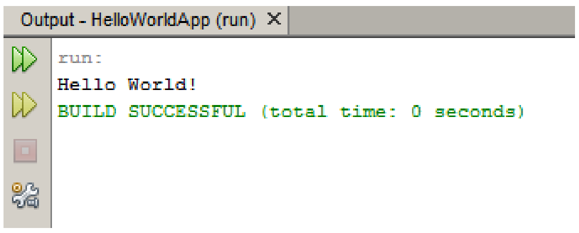
- nbproject: this contains a variety of files which the IDE uses to manage the building process and also to maintain other environment information relating to the project.
- src: this folder contains the source code files (.java) for the java classes making up the application. In this case we only have a single class and it is thus stored in a single package within the source folder. As indicated, more complete java applications/projects are generally structured as a collection of separate packages with each package containing one or more classes making up the overall application. Structuring a set of classes into packages enables the isolation of related components so that these can be easily reused in other applications.

- *Depending on the configuration of your IDE when set up, you might also see another folder, i.e. test: this folder will contain the test classes used to test the code if you are using an automated testing framework as part of your development. We will be using this with the JUnit automated testing framework from the week 2 lab exercises.*

You should note, however, that there is no "object" code (i.e. there is no helloworldapp.class file). That is because we have not yet compiled (i.e. built) the program. Thus we shall now (whilst still having the Files tab selected) run the program. To do this:

- Choose **Run -> Run Project** (or just press the green "play" button on the Quick Access toolbar at the top of the IDE window).

You should now see the following (in the Output pane) below the Editor window.



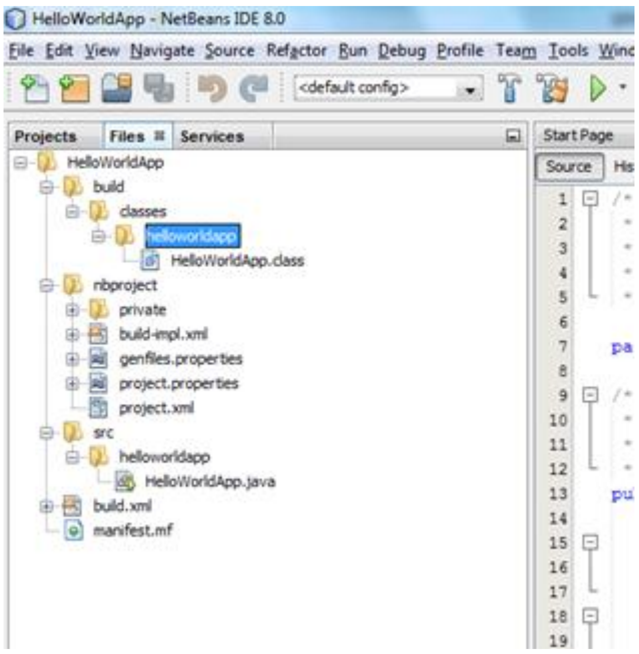
```
run:
Hello World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Congratulations! Your program works!

If there are compilation errors, they are marked with red glyphs in the left and right margins of the Source Editor (and a red squiggly line under the code itself). The glyphs in the left margin indicate errors for the corresponding lines. The glyphs in the right margin show all of the areas of the file that have errors, including errors in lines that are not visible. You can move the mouse over an error to get a description of the error. You can click a glyph in the right margin to jump to the line with the error.

As well as noting your program has worked correctly, you should also now note that a new folder (named **build**) has now appeared under the project's file structure.

Fully expand this folder (and its subfolders) and you should get something similar to:


HelloWorldApp - NetBeans IDE 8.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Wind
<default config>
Projects
Files
Services
Start Page
Source
His

```
HelloWorldApp
  build
    classes
      helloworldapp
        HelloWorldApp.class
  nbproject
    private
    build-impl.xml
    genfiles.properties
    project.properties
    project.xml
  src
    helloworldapp
      HelloWorldApp.java
  build.xml
  manifest.mf
```

What you now see here is a folder called **build:** this folder contains the **.class** files produced by the java compiler, i.e. the “object” code (in java’s case, java bytecode format) which can then be executed.

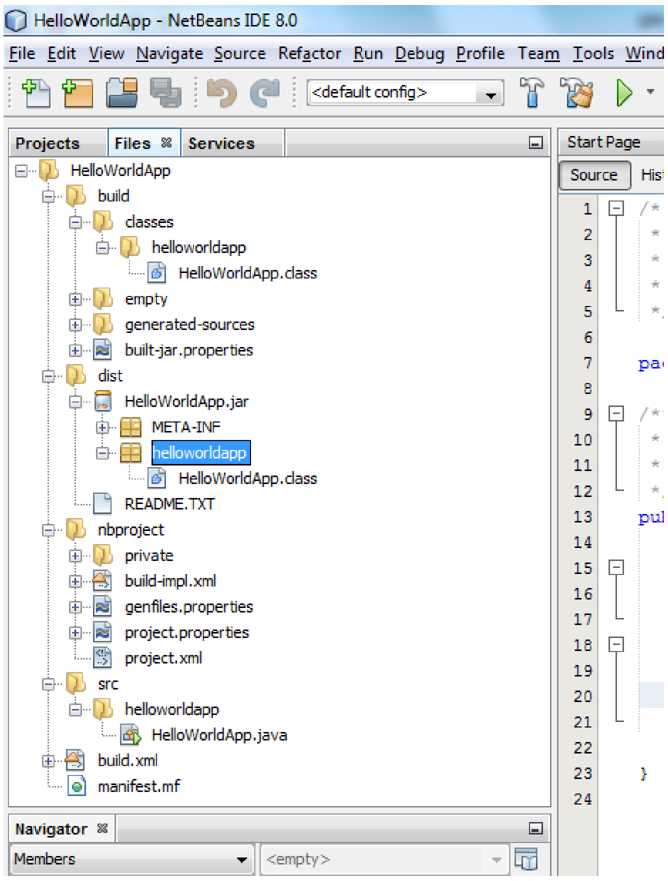
If we then go further, and "fully build" this project, by (whilst you still have the File tab selected) thus:

- Selecting **Run -> Clean and Build Project**

Once this has completed (after a few moments) you will see that the build folder has additional files/folders and that there is also another project level folder named **dist**.

- Expand this **dist** folder and then expand the **HelloWorldApp.jar** item which appears and then expand the **helloworldapp** package item, which subsequently appears.

Your Netbeans screen should now look similar to below:



In terms of contents:

- dist: this folder contains the distribution file(s) for the java project. Most programs are modularised and contain (at least) several source files with different components. After a project build, the separate object code files need to be linked together with some mechanism to be executable as a "single" application. In java a **.jar** file is a **J**ava **AR**chive file, which is a collection of java code files for an application/software component. These can be directly executed using the Java Virtual Machine. Obviously in this case, our application consists simply of one "code" file and so the contents of this JAR file are not particularly complex.

The overall purpose of JAR files is to enable the easy distribution and use of the java "object" code components. If, for example, one developer writes some code classes, which another developer might wish to use, then all that needs to be supplied to the other developer is the (executable) JAR file for the code/classes. The original source code for the classes is not thus required. Then all the developer has to do is to "import" the JAR file into their project and the project can then use those classes.

Executable JAR files are also "stand alone" executable code and thus can be executed independently of the IDE. After all an IDE is used to develop software/programs, it is not required by a user of the finished software product! The user only needs the "fully packaged" executable ("object") code for an application properly installed on their PC. Thus we shall see in the next exercise how this JAR file can be executed on a "stand alone" basis.

Ex 2: Executing a JAR file as a "stand alone" program

In order to best illustrate this, we shall first enhance our program (from Ex 1) by making it output the Hello World greeting on a Dialog Box rather than as plain text output on the console window/IDE Output pane. Thus (after reselecting the **Projects** tab on the IDE):

- Amend the **HelloWorldApp.java** file's contents so that the source text is as follows (for easy of understanding the changes to the text are highlighted):

```
package helloworldapp

import javax.swing.JOptionPane;

public class HelloWorldApp
{
      public static void main (String[] args)
      {
            //Display a greeting in a dialog box
            JOptionPane.showMessageDialog(null,"Hello World");
      }
}
```

This version uses a java class library (there are literally hundreds of java class libraries in the java API) which provides object based facilities for windows components. After making these changes:

- Re-run the project, via **Run -> Run Project**

This time the program will display the message on a simple dialog box, you should then:

- Click **OK** on that box to end the program.
- Then, perform a (Run ->) **Clean and Build** to ensure that the **JAR** file (for distribution) is updated.
- Then close the Netbeans IDE (via **File -> Exit**)

This amendment has also illustrated another important aspect of software development. That is the use of classes and associated program components written by other developers. The `JOptionPane` class is not formally part of the core java language. It is part of the extensive (and extensible) library of java classes provided along with the Java JDK. In java a convenient mechanism for collecting a set of related classes is the use of a java package (the java package **javax.swing** contains a number of useful classes which provides windows components, including **JOptionPane** which is the class which we are using here to create a dialog box). Thus if we want to use it we must "import" it (or rather its interface) into our program by using the `import` statement (i.e. `import javax.swing.JOptionPane`) in our program. You should note that `import` statements are placed "outside" of the class definition (but "inside" the package definition) at the start of any java program file.

- Having closed the Netbeans IDE, launch **Windows Explorer** and navigate to the **dist** folder within the **HelloWorldApp** folder for the project. Normally Netbeans Projects are within **(My) Documents -> NetBeansProjects.**

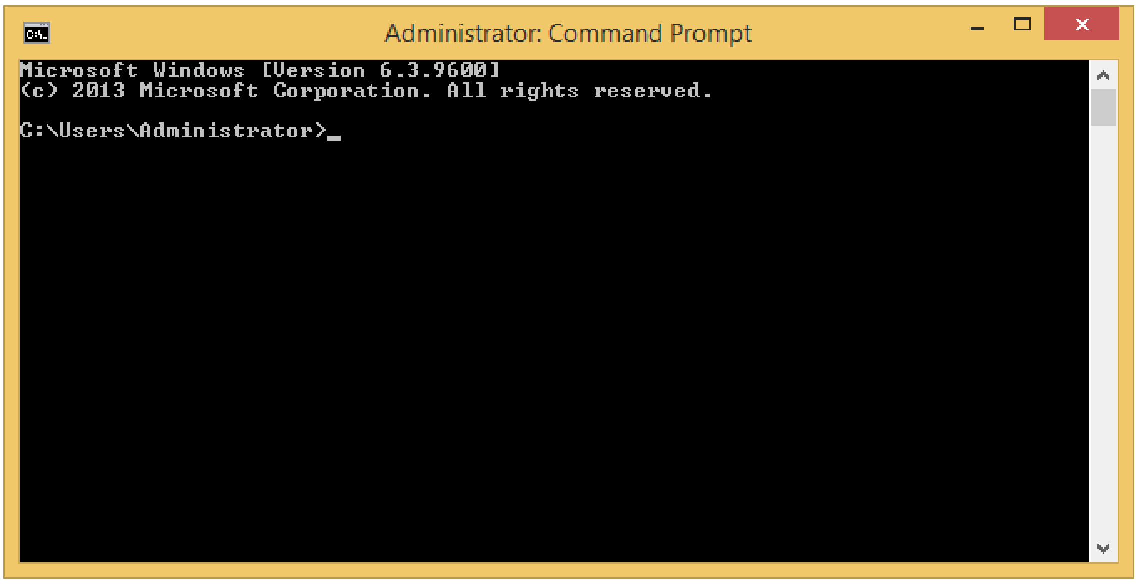
You should see the **JAR** file **HelloWorldApp.jar** (its icon should be the little java platform symbol). Then:

- Execute this JAR file by doubled-clicking on the file/icon
  (or via **Right-Click -> Open**)

After a few moments the program should execute (and you will see the dialog box). This execution can also be done from the MS-DOS Console/Command Line. To do it that way, simply:

- Select **Start -> All Programs -> Windows System -> Command Prompt OR**
- Select the Command Prompt icon found on the bottom task bar

The console window should be similar to below;



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>_
```

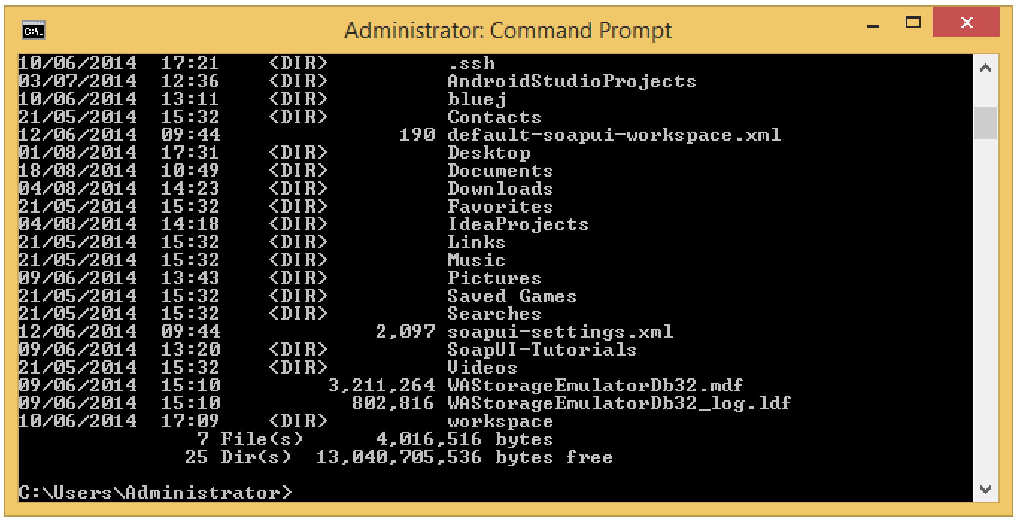
The default directory may be different from above, depending upon the initial configuration of the machine you are using. It certainly will be different if you are doing this on your home computer.

This is the style of interface PCs had before Graphical User Interfaces (i.e. the sort of interface a "dinosaur" like me was "brought up on" at I was at University)! From this window you can enter operating system commands directly from the prompt (these commands are the underlying Microsoft Disk Operating System – MS-DOS – commands which are still part of the Windows operating system). The prompt on the command line ends with > and always has the full pathname of the current directory.

Thus if you type the command **dir** from that prompt (**dir** is the MS-DOS command to list the contents of the current directory), then you will get a listing of the contents of this **C:\Users\Administrator** directory. (Remember that if your command prompt is not set to that C:\Users\Administrator directory then you will have to explicitly change to it first by typing **cd c:\users\administrator** for the command prompt.) Thus:

- Enter the command **dir** at the command prompt

You should then see a (probably quite lengthy) listing of that directory similar to:



```
10/06/2014  17:21    <DIR>          .ssh
03/07/2014  12:36    <DIR>          AndroidStudioProjects
10/06/2014  13:11    <DIR>          bluej
21/05/2014  15:32    <DIR>          Contacts
12/06/2014  09:44               190 default-soapui-workspace.xml
01/08/2014  17:31    <DIR>          Desktop
18/08/2014  10:49    <DIR>          Documents
04/08/2014  14:23    <DIR>          Downloads
21/05/2014  15:32    <DIR>          Favorites
04/08/2014  14:18    <DIR>          IdeaProjects
21/05/2014  15:32    <DIR>          Links
21/05/2014  15:32    <DIR>          Music
09/06/2014  13:43    <DIR>          Pictures
21/05/2014  15:32    <DIR>          Saved Games
21/05/2014  15:32    <DIR>          Searches
12/06/2014  09:44             2,097 soapui-settings.xml
09/06/2014  13:20    <DIR>          SoapUI-Tutorials
21/05/2014  15:32    <DIR>          Videos
09/06/2014  15:10         3,211,264 WAStorageEmulatorDb32.mdf
09/06/2014  15:10           802,816 WAStorageEmulatorDb32_log.ldf
10/06/2014  17:09    <DIR>          workspace
               7 File(s)      4,016,516 bytes
              25 Dir(s)  13,040,705,536 bytes free

C:\Users\Administrator>
```

Obviously to be able to run the HelloWorldApp.jar file we need to get to the **dist** directory within the NetBeans Project, within the overall C: file system. To do this we have to change the default directory using the MS-DOS command **cd** (which stands for “change directory” - this will change the current directory to a named directory).

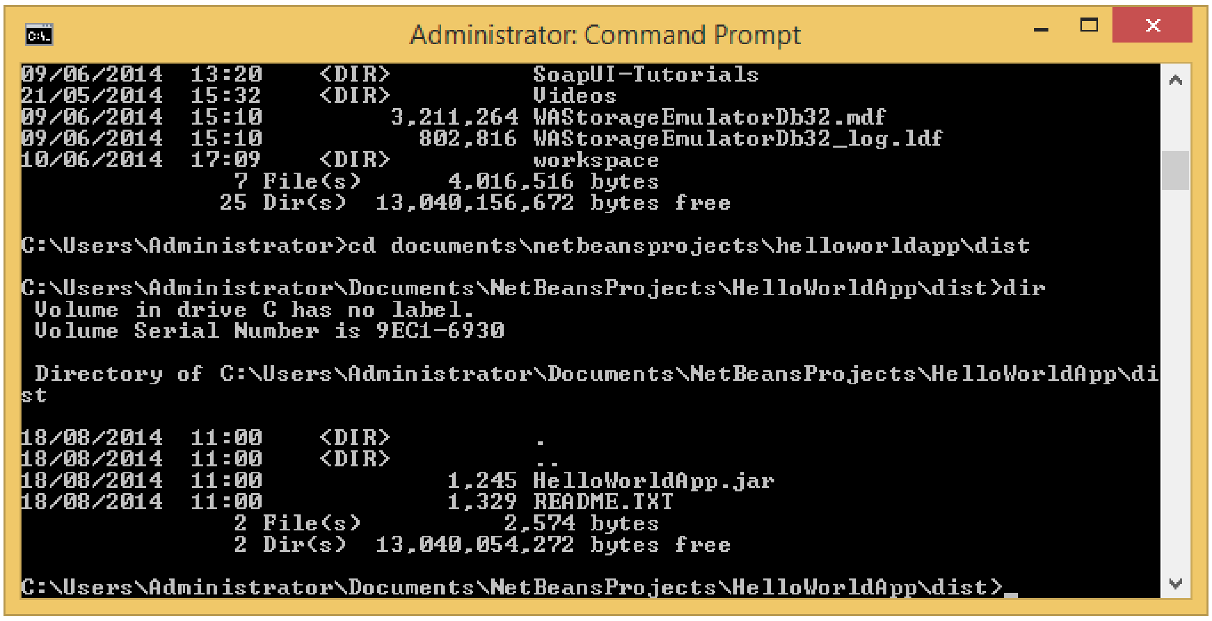
To get to the necessary folder, we would need to successively use the cd command from the > prompt to navigate to that folder. You should be able to see that the **(My) Documents** folder (in which all of the NetBeans projects are located) is “inside” this initial **Users\Administrator** directory. You could do this by initially typing cd users and then using **dir** to then see the next directory and successively navigating, or if (as here) you know the full pathname, you can do that in a single (rather lengthy) **cd** command. We will first “have a go” with the full pathname from Users\Administrator, thus:

- Type **cd users\admininstrator\documents\netbeansprojects\helloworldapp\dist** from the > prompt.

Hopefully, if this works, (you will know that because it should then give the full pathname of the current directory as part of the > prompt) if you then:

- Type **dir**

You will see that the **HelloWorldApp.jar** file is there. The console window should look similar to below, if you have done the set of commands correctly:



```
09/06/2014  13:20    <DIR>          SoapUI-Tutorials
21/05/2014  15:32    <DIR>          Videos
09/06/2014  15:10         3,211,264 WAStorageEmulatorDb32.mdf
09/06/2014  15:10           802,816 WAStorageEmulatorDb32_log.ldf
10/06/2014  17:09    <DIR>          workspace
               7 File(s)      4,016,516 bytes
              25 Dir(s)  13,040,156,672 bytes free

C:\Users\Administrator>cd documents\netbeansprojects\helloworldapp\dist

C:\Users\Administrator\Documents\NetBeansProjects\HelloWorldApp\dist>dir
 Volume in drive C has no label.
 Volume Serial Number is 9EC1-6930

 Directory of C:\Users\Administrator\Documents\NetBeansProjects\HelloWorldApp\di
st

18/08/2014  11:00    <DIR>          .
18/08/2014  11:00    <DIR>          ..
18/08/2014  11:00             1,245 HelloWorldApp.jar
18/08/2014  11:00             1,329 README.TXT
               2 File(s)          2,574 bytes
               2 Dir(s)  13,040,054,272 bytes free

C:\Users\Administrator\Documents\NetBeansProjects\HelloWorldApp\dist>_
```

Then to run this file:

- Type **java –jar helloworldapp.jar** from the prompt

After a few moments the dialog box message will come up as the program executes.

<u>Saving your work at the end of a VMware Session</u>

Since, in the University labs, we are using a Virtual Machine to provide the environment for our java application development, then the work you have created only exists on the virtual hard disk within the Virtual Machine "window". None of it is being saved onto the actual PC's hard disk. Thus when you close down that virtual machine (window) all of your work will be lost! Thus before you close down your VMware player window you will want to save the work you have done. To do this you have a couple of options:

- The simplest option is probably to plug in a USB drive to the host desktop, then to go to the VMWare player and (using Windows Explorer) find the NetBeans Project folder and copy the appropriate project folder (by using **Right Click -> Copy**). Then return to the host Desktop and then paste (via

Right Click -> Paste) the folder either to the desktop/or directly to the USB drive. **OR**

- If you plug in a USB drive when input is directed to the VMPlayer window then USB drive should be connected to the VMPlayer. After that if you use Windows Explorer within the VMPlayer then you will see the USB mounted and you can drag and drop as required. In this case you should notice that on the title bar of the VMPlayer window there is a menu option (which should be under) **Player -> Removable Devices** and this will also "see" the USB drive if it is mounted. (In practice I have found that this option is rather problematic as the VMware Player does not consistently mount the USB drive). **OR**

- You can just zip up the NetBeans folder (under **(My) Documents -> NetBeansProjects**, by right-clicking the project folder **->** **Send to -> Compressed (Zip) folder**) and then emailing it to yourself using a browser from the Virtual Machine.

Thus you should copy your work onto a USB device (or zip up the project and email it to yourself) so that you have a copy for your own further reference and study.

Downloading Netbeans for use on your Home PC

Remember that you will no doubt wish to download the Netbeans/Java environment for yourself for use on your Home PC/laptop. You may not yet have done this, preferring instead to wait until you had completed the first weekly exercise before doing this. However, now that you have completed this exercise you may wish to return to the previous instructions (Downloading Netbeans for use on your Home PC, on page 6) to download and install the Netbeans/Java environment at your convenience.

Week 2: Testing a class with a test "harness" and an introduction to the JUnit automated Test Framework

The lab exercises in this second week are aimed at enabling you to start to develop an understanding of the following objectives of the lab work:

1. Testing (java) components through the construction of simple test "harnesses"
2. Using the JUnit test framework to automate the testing process

Remember again that these exercises, when undertaken in a University Lab, must be done from within the VMware Java Apps environment.

There are 3 files needed in this lab. Download them from GCU Learn/BlackBoard first as follows:

- Using any appropriate browser in the VMware image:
  - Login to **GCU Learn/Blackboard**[6] **and go to the module:** 14/15 A - SOFTWARE PROCESSES AND PRACTICES
- Then from **Learning Resources -> Lab Work -> Week 2 Lab** download the following three files to the **Desktop** of your VMware image:
  BankAccount.txt,
  BankAccountTest-for-Netbeans.txt and
  BankAccount-TDD-Initial-Skeleton.txt

Ex. 1: Developing a java application which consists of more than one class

A typical java application will actually consist of a number (often a very large number) of classes. Even this was demonstrated with the Week 1 dialog box based **HelloWorldApp** since the `HelloWorldClass` used "code" from another class (i.e. the `JOptionPane` class). Classes in any modern programming language are used as individual "building blocks". Each class will generally represent a distinct "Real world" object/software component and provide the functionality for that object.

For example a bank application will require a class which represents a bank account. That `BankAccount` "object" will require holding certain property variables/instances. It will also have a set of methods which initialise the class instance and also enable the set of banking transactions to be carried out on it. However, the `BankAccount` object itself will not have a (`public static`) `main` class. Only the overall bank application controlling class which initialises the application and then sets off it various functions will have a `main` class. Thus the class with the `main` method will need to be "linked" to the `BankAccount` class within its code. This next example will be a very simplified example which involves a program across two separate files (and classes). One class is a (very simplistic) `BankAccount` class, and the other class is called `BankAccountTest` and it has a `main` method. The purpose of the `BankAccountTest` class is to run some code which enables the functionality of the `BankAccount` class to be tested. However, rather than start by giving the source code for both classes, we shall start with a description of the functionality of the `BankAccount` class.

[6] If GCU Learn/Blackboard is not yet available to you due to registration problems, ask the lab tutor for copies, or get them from a fellow student.

The class `BankAccount` contains:

The following property:

- `balance`: of type `double` which contains the account's monetary balance

The following methods:

- `BankAccount`: the class Constructor which takes a single parameter `amount` (of type `double`) which is set as the opening balance of the account.
- `getBalance`: which returns a `double` value which is the current balance of the account.
- `withdrawMoney`: which takes a single parameter `amount` (of type `double`) and deducts the amount from the balance of the account
- `depositMoney`: which takes a single parameter `amount` (of type `double`) and adds the amount to the balance of the account

This, the definition of the interface of the class, is actually very important design information. It is important for software quality that a software component is clearly specified before any code for it is actually written. A key premise of Agile Methods, Test Driven Development and good software development generally is that the specification of a module/component should be determined before any code for it is written! Many errors in the software development process are due to incorrectly specified software or software components that do not integrate correctly. Anyway the simplified code for a version of this `BankAccount` class is given below:

```
public class BankAccount {

       private double balance;
       //in a "real life" application this class would have more
       //properties for customer info etc. as well as more methods

       public BankAccount(double openingBalance)
       {
           balance=openingBalance;
       }

       public void withdrawMoney(double amount)
       {
          balance=balance-amount;
       }

       public void depositMoney(double amount)
       {
           balance=balance+amount;
       }

       public double getBalance()
       {
           return balance;
       }
}
```

Rather than getting you to type this in yourself, one of the files supplied is a textfile called **BankAccount.txt**. Thus, you should now do the following tasks in Netbeans:

- Using **File -> New Project**, create a java project called **BankAccountApp**, for the moment ensure that the **Create Main Class** is checked and entered as **bankaccountapp.BankAccountApp**.
- In the subsequently created project expand the **Source Packages** and immediately delete (via right click) the created class skeleton file **BankAccountApp.java**
- Create a New Class, called **BankAccount.java**, by right-clicking on the Source Package (**bankaccountapp**) and selecting **New -> Java Class**, then entering the class name as **BankAccount** in the subsequent dialog box.
- Now from the VMware desktop, open up using **Notepad** or **TextPad**, the supplied file **BankAccount.txt**, and (via cut and paste) select the complete text of that file and replace the text of the skeleton class in the NetBeans source editor window for that class with these contents.
- Try to **Run** this program and you will see that you cannot! (An error dialog box appears telling you that you do not have a main class in this project. Dismiss this box, via Cancel, once you get this.)

Of course this is precisely what was expected (wasn't it?), and demonstrates a dilemma which has to be resolved in a modern software development environment. That is, a complete program may consist of many different classes across many different files. However, none of these files will have a Main class/main method. Only one file will in a complete collection of the files making up a complete application will have a "main" class with the `public static void main` method. Thus how can we test the individual component before trying to integrate it with the larger software application which is being developed? If you do the following:

- Select **Run -> Build Project**

Then you will see (if you go to the **Files** tab once this has completed) and expand the **build** and **dist** folders, that the **BankAccount.class** "object" code and the **JAR** file for the code has been created. However, again, we cannot actually run this class to enable us to test it. Thus what will we have to do?

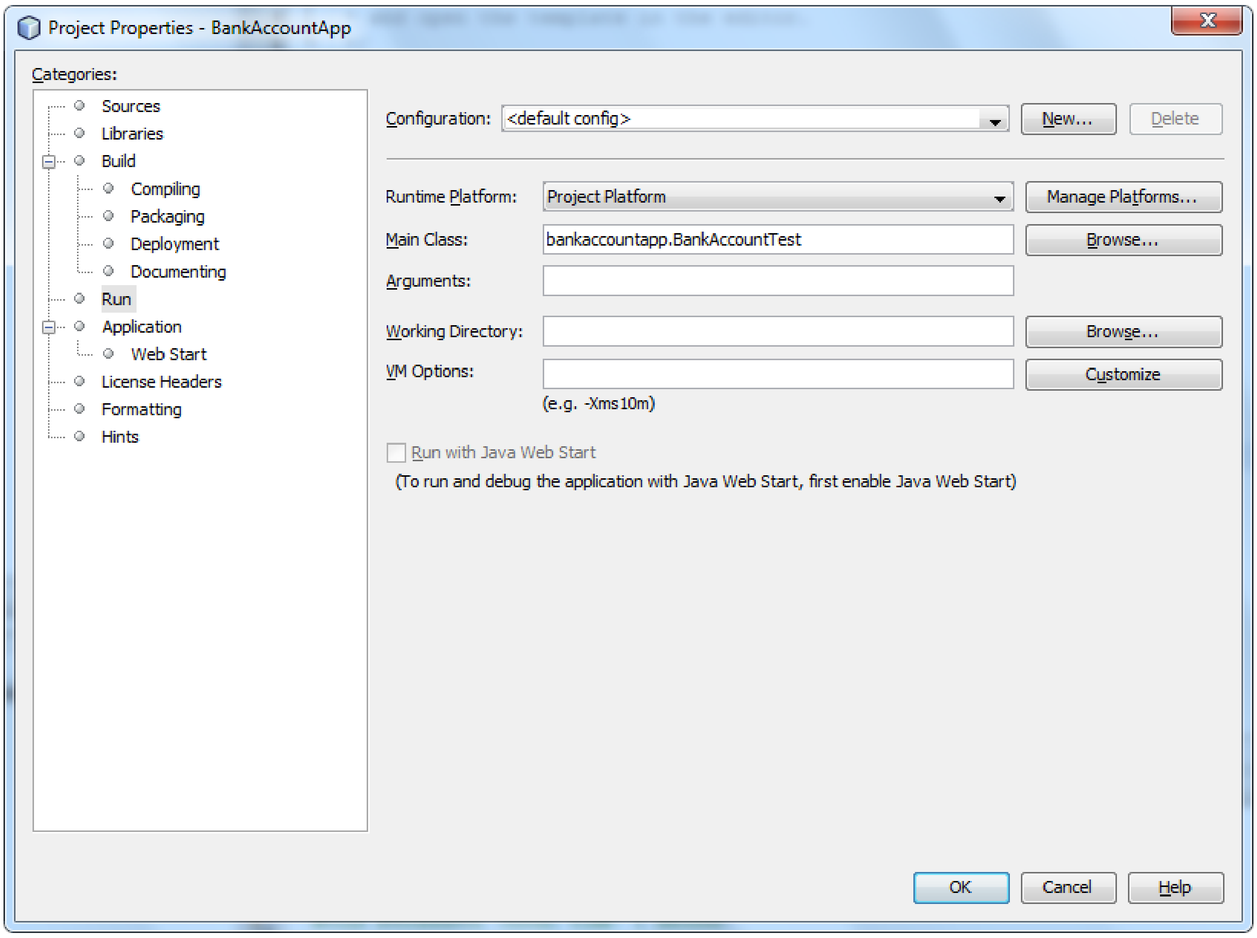
The answer is simple, but "laborious". We will have to add another class to this program, which is/has a Main class/method and write "test harness" code in that class to create and execute, and check that the BankAccount class is working correctly. Thus we are now going to do precisely that.

We are going to create the following class (called BankAccountTest) with the following code (OVERLEAF):

```
public class BankAccountTest{

      //This program will act as a test harness for BankAccount.java
      //It will set up a class instance and execute the methods with
      //known values and output the results to the "screen" to allow
      //the tester to see the results and determine whether the
      //BankAccount class has been properly implemented

      //To make the test results clear, the code will "test" the
      //returned results and put explicit output messages to clearly
      //show whether the test PASSED or FAILED. In the case of a FAIL
      //message the value of the appropriate returned variable will
      //be displayed to enable the developer to have useful debug
      //information to help in identifying the possible problem

      public static void main(String[] args)
      {

         //Declare a new BankAccount class instance to use in the
         //test and set it with an initial balance, say £500
         BankAccount testAccount = new BankAccount(500);
         double currentBalance;

         //Test the constructor for setting the balance correctly
         currentBalance = testAccount.getBalance();
         if (currentBalance == 500)
            System.out.println("Testing Constructor - PASSED");
         else {
            System.out.println("Testing Constructor - FAILED");
            System.out.println("Actual Balance was:"+currentBalance);
         }

         //Test the withdrawMoney method by withdrawing an amount
         // from the account and then checking to see if the balance
         // has been correctly updated
         testAccount.withdrawMoney(200);
         currentBalance = testAccount.getBalance();
         if (currentBalance == 300)
            System.out.println("Testing withdrawMoney - PASSED");
         else {
            System.out.println("Testing withdrawMoney - FAILED");
            System.out.println("Actual Balance was:"+currentBalance);
         }


         //Test the depositMoney method by depositing an amount from
         //the account and then checking to see if the balance has
         //correctly been updated
         testAccount.depositMoney(100);
         currentBalance = testAccount.getBalance();
         if (currentBalance == 400)
            System.out.println("Testing depositMoney - PASSED");
         else {
            System.out.println("Testing depositMoney - FAILED");
            System.out.println("Actual Balance was:"+currentBalance);
         }

      }
}
```

Study the above code carefully and then do the following (**OVER PAGE**):

- Add a(nother) new class called **BankAccountTest** to the Source Package (bankaccountapp). Cut and paste its contents from the supplied file: **BankAccountTest-for-Netbeans.txt**
- Set the main class of the project to **bankaccountapp.BankAccountTest** by selecting **File -> Project Properties**, then selecting **Run** from the Categories: and then changing the **Main Class:** text box entry to **bankaccountapp.BankAccountTest**. (as per the screenshot below)



Configuration: <default config>

Runtime Platform: Project Platform

Main Class: bankaccountapp.BankAccountTest

Arguments:

Working Directory:

VM Options:

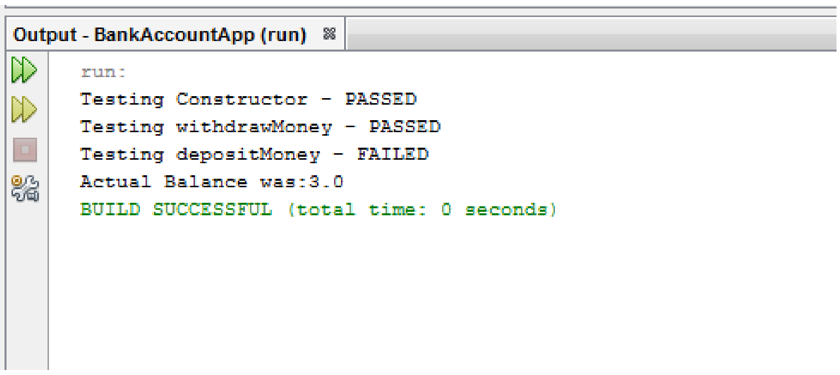
(e.g. -Xms10m)

Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)

- Click **OK** to dismiss this dialog and then run the program.

What did you get? If you have done all of the tasks as instructed, then your output should be as below:



```
run:
Testing Constructor - PASSED
Testing withdrawMoney - PASSED
Testing depositMoney - FAILED
Actual Balance was:3.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Is this the output you got? Is it correct? Does it make sense? Does it indicate that the **BankAccount.java** source code which I supplied to you, in the textfile, was correct?

Thus you should be able to see by analysing the output for yourself, that the output for the execution of the constructor and the `withdrawMoney` methods are okay but that the observed output after the `depositMoney` method has executed is not as you expected the output to be. Since I gave you the file **BankAccount.txt** (which supposedly contained the "correct" code) you probably didn't check to see if its contents matched **exactly** the text of the correct source code for the `BankAccount` class which was given in Page 24 (above)! Thus:

- Open up on the Netbeans source code editor the file **BankAccount.java** and look very carefully at the code for the `depositMoney` method of the class `BankAccount`. Does it match the code given on Page 24 (i.e. does it **add** the `amount` to the `balance`)?

Of course, the answer should be that it doesn't and there has been a simple (if deliberate on my part) typo and I have divided indeed of added! An easy mistake which anyone might actually make when writing code! Thus:

- Change this line to: `balance=balance+amount;`
- Re **run** the program

This time, the output should indicate that all tests have successfully passed.

What you have actually done in this exercise is undertake a simple Unit Test. Unit Testing is a very important part of the overall testing process. Since the "lower level" units in a program, i.e. the individual class components of an application cannot execute on a "stand alone" basis, then additional software components (known as either Test Harnesses, Test Fixtures or Test Scaffolding) must be written to test the functionality of the class. This has to be done in the local workspace of the developer who is working on the class. This testing must be carried out before the class is deposited onto any central repository and integrated into any other developed software components of the application. If no testing is done on individual components before they are integrated then the developers will probably get lots of errors, all from different components of the application. However, generally because there are so many errors which all appear at the same time (i.e. at integration) then it becomes a very difficult process to isolate and correct each individual error. It is far better to test each individual component as a "stand alone" so that most (if not all) of the errors within the module/class have been resolved before that module is integrated into the rest of the developed code.

One thing you may have noticed about the output is that you have to "manually" i.e. "by eye", check that the outputs of the tests give the expected results. This process is rather "labour intensive" and potentially quite error prone itself and the tester/developer could easily "miss" a failed test. Obviously I have tried to make it "easier" to spot including code in the test harness which "physically" tested to see if the observed output was the same as the expected output and also including details of which method was being tested by each output line, and the actual output of the key variable should the test fail.

After having undertaken this exercise, you might probably be thinking – “Isn’t this process of testing a time-consuming, quite intricate and labour intensive exercise for the developer?” Indeed, if that is what you are thinking, then you are absolutely correct! Possibly it is for reasons such as this that developers (and projects) at times often “skimp” the amount of testing undertaken as pressure increases for delivery. This is actually a false economy. Thus what really has to be done is to find ways of automating and/or making the process of checking test output easier. Essentially the process with this NetBeans Exercise has been to have the “test” class (i.e. **BankAccountTest.java**) incorporate code to execute each method for already known values and then to test the observed output against those. The automated testing framework, known as JUnit provides facilities for just that, so that the “labour” can be taken out of the process.

Ex. 2: First Introduction to the JUnit Automated Testing Framework

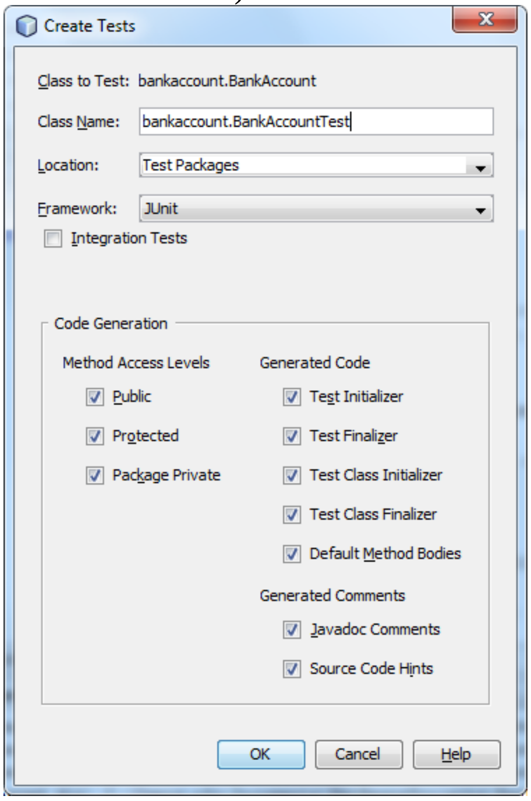
JUnit is part of the xUnit framework of classes. In order to successfully undertake efficient Unit testing, and in particular to undertake Test Driven Development (TDD), the testing process should be as automated as possible. Thus JUnit (which is the java version of the xUnit framework) contains a set of classes and facilities to enable automated running of unit tests. The details of the framework will be given within the related lectures on this module, but for the time being we will see in this exercise how the JUnit framework is integrated into the Netbeans IDE. For the moment all you need to appreciate is that the basic principle of using JUnit is to develop a “test” class for every class you develop (and thus require to test) and within that class a “test” method for each method within the class which you wish to test. This is much the same idea as the “manual” exercises we have just undertaken.

Thus in the “manual” example, we had 4 methods (including the Constructor and the “getter” method, `getBalance()`) and had 3 “explicit” tests. The only method we didn’t “explicitly” test was the `getBalance()`. This was because it was used to get the results of the other methods’ tests. Thus you will see that in many cases, getter methods are “implicitly” tested as “by products”.

In TDD the principle is that tests should be developed before the code is “written”. In reality that often means that developers write the interface of the class and the class’s methods as each test is developed, but only write some “dummy” or “stub” code in those class methods of the class being developed. This is so that the specification of the interface is confirmed (as this is very often a source of errors in a development) and also to ensure that the tests, when initially written and run, are all forced to fail. Remember that a key principle of TDD (and the Agile approach generally) is that all tests must first be developed and shown to fail, then as the production code is developed and each test passes then the developer is clear that sufficient code to implement each function has been done. Then when all tests pass, the developer knows that no more code need be written to implement the code’s functionality any more. However in TDD/Agile further development is carried out to refactor the code, i.e. to optimise and improve its internal quality, but NOT to add or change any functionality. This is often called the RED-GREEN-GREEN approach (for reasons which will become apparent when we implement our first JUnit based test set in Netbeans.

In Netbeans, there is a wizard to help developers to develop tests. With Netbeans, the developer would construct the skeleton (partial or otherwise) of the class being developed. This development code is placed in the **src** folder for the project (as usual). The wizard, when invoked, will place a skeleton test class in the **test** folder of the project. Anyway let's get started with this introductory exercise. In this we will create a project which only consists of the `BankAccount` class and then we will test it using the integrated facilities provided by Netbeans for the use of JUnit.

- Launch Netbeans and using the **File -> New Project** wizard create a new Netbeans project called **BankAccount**.
- Once created, replace the skeleton content of the created BankAccount class by pasting in the source text from the file **BankAccount-TDD-Initial-Skeleton.txt** (the last of the 3 files supplied to you for this lab). Remember not to delete the first line `package bankaccount;` from the initial skeleton. Then **Build** (<u>but don't run</u>) the project. The project should compile cleanly.
- Study the source code of the `BankAccount` class you have just created. You should note that its contents are essentially dummies/stubs for the key methods (except the "getter" method which is needed by the tests to "expose" the results)
- Then right click on **BankAccount.java** in the Project windows and then select **Tools -> Create/Update Tests** from the menu. You should then get a dialog box confirming the that a set of JUnit Tests (to be placed in a test class called BankAccountTest) will be created. The dialog box is given below.


Create Tests

Class to Test: bankaccount.BankAccount

Class Name: bankaccount.BankAccountTest

Location: Test Packages

Framework: JUnit

☐ Integration Tests

Code Generation

Method Access Levels
☑ Public
☑ Protected
☑ Package Private

Generated Code
☑ Test Initializer
☑ Test Finalizer
☑ Test Class Initializer
☑ Test Class Finalizer
☑ Default Method Bodies

Generated Comments
☑ Javadoc Comments
☑ Source Code Hints

- Select OK. You will then be asked which version of JUnit to use, select version **JUnit 3.x** (note, that it is <u>**not**</u> the default) and then press the **Select** button.

This will have created a skeleton test class for the project's `BankAccount` class. The test class will be called `BankAccountTest` (the standard convention when using JUnit, i.e. name of original class but with the text "`Test`" added). It should be opened automatically in the right hand edit pane, but if you fully expand the **Test Packages** folder of the project you will see where it has been created (i.e. in the **test** folder of the project). The test class skeleton created has some dummy code in it which ensures that the tests will fail (although it is not sufficient for our purposes). If you study that test class then you will see that it includes the following (in "top to bottom" order):

- An `import` statement for access to the JUnit framework and its functionality.
- The declaration of the test class `BankAccountTest`. You should note that it inherits from the JUnit framework class `TestCase`. This parent class provides the core automated execution facilities for running the tests.
- A constructor for the test class, which at present simply initialises itself by calling the parent class constructor (this is what the method `super()` does)
- Two methods `setUp()` and `tearDown()`. These are two methods which JUnit always calls immediately before and immediately after executing each test method in a test class. These are use to initialise any "state" needed for a test and to clean up that "state" afterwards. We will not initially be making use of these facilities just yet in this starter exercise.
- 3 test methods (`testWithdrawMoney()`, `testDepositMoney()`, `testGetBalance()`) for each of the public methods we have in our class. Note that the wizard does not create skeleton test methods for the class constructor since these are not always explicitly tested, but are often implicitly tested). The convention which the JUnit (version 3.x) framework expects is that the name of each test method starts with the text "test". Thus on execution, the framework essentially "scans" the test class for each method name starting with "test" and executes each in turn. A simple idea really!

To develop our complete test class for this first exercise, do the following:

- Delete the text for the 3 skeleton test methods `testWithdrawMoney()`, `testDepositMoney()` and `testGetBalance()` and put in the following methods instead:

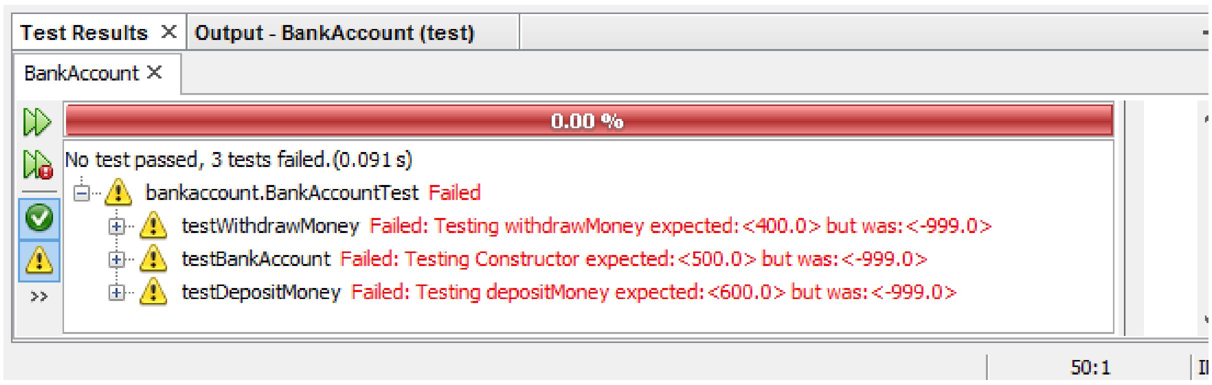
```
public void testBankAccount() {
  BankAccount testAccount = new BankAccount(500);
  assertEquals("Testing Constructor",500.0, testAccount.getBalance());
}

public void testWithdrawMoney() {
    BankAccount testAccount = new BankAccount(500);
    testAccount.withdrawMoney(100);
    assertEquals("Testing withdrawMoney", 400.0, testAccount.getBalance());
}

public void testDepositMoney() {
    BankAccount testAccount = new BankAccount(500);
    testAccount.depositMoney(100);
    assertEquals("Testing depositMoney", 600.0, testAccount.getBalance());
}
```

The code in each of these three test methods makes use of one of the core features of (JUnit) testing. That is, the use of an `assert` statement (in this case a version of the `assertEquals()` statement). In a similar fashion to the `if` statement used in the manual version developed earlier, this statement checks to see if the values given are "equal" (i.e. as expected by the test). Thus you can see (without understanding too much at present about the "detailed workings" of JUnit) that we have 3 tests, each which starts by setting up the required "state" for the test, executing the necessary method to be tested and then checking to see if the method has produced the expected output for that particular test. Now do the following:

- Run the tests by right clicking on the project name and selecting **Test** from the menu. This will compile all the files (including the test class) and then run the test class. Remember that if you have any compiler errors these will be highlighted in the Output pane (and also as a squiggly red line in the edit window) and Netbeans will not then run the executable code. It will be highly likely that you will have made some typing errors in trying to recreate the Test Code and so make the corrections until you successful run the tests.

When you do successfully compile and run the tests, this time Netbeans produces a (GUI based) **JUnit Test Results** pane and in it you should note that the outcome is that the test class has failed.

- Expand this test output result, by clicking on the small plus sign icon to the left of the yellow exclamation mark symbol. Also increase the size of the test results output window so that you can fully see the various messages. You should get something similar to below (possibly the test methods may have been executed in a different order):



```
No test passed, 3 tests failed.(0.091 s)
bankaccount.BankAccountTest Failed
   testWithdrawMoney Failed: Testing withdrawMoney expected:<400.0> but was:<-999.0>
   testBankAccount Failed: Testing Constructor expected:<500.0> but was:<-999.0>
   testDepositMoney Failed: Testing depositMoney expected:<600.0> but was:<-999.0>
```

Thus you should see each test method's FAILED node. You will see that it gives the reason for each fail, indicating clearly both the expected and observed output.

Now let us try to get the tests to pass.

- Go back to the **BankAccount.java** source code and recode the Constructor correctly, i.e. replace its code with `balance=openingBalance;`
- Re-run the tests (by right clicking on the project name and selecting **Test**)and note that whilst the overall tests fail, the constructor has PASSED (and is shown as green) and the top bar of the Test Results pane shows the percentage progress of passing tests (also in green)

- Select the **Output – BankAccount (test)** pane and you will also see the text output version of the tests which have just been run.
- Enter correct source code for each of the other two methods of the class `BankAccount`, re-running the tests each time, i.e.
    - `balance=balance-amount;` (for withdrawMoney()), and
    - `balance=balance+amount;` (for depositMoney())

Well done, if you get all of the test to "go green", you have now successfully conducted your first exercise in using JUnit with TDD.

Ex. 3: Additional exercise to undertake

When running a set of tests, it is good practise to make sure that each test does not depend upon a previous test. If you recall in the initial (Command Prompt and NetBeans) examples our initial (Constructor) test set the opening balance to £500, but then each of the next two tests depended upon the output from the previous test being successful, i.e. `withdrawMoney()` test presumed the balance of £500 was okay from the Constructor test and then the `depositMoney()` test presumed that the balance from the `withdrawMoney()` test was okay (i.e. was now £300). However, if any of the previous tests failed then this would have had a “knock on” effect on the subsequent testing. Indeed what it would do is give a “false impression” that a method failed, when it may actually be correctly coded. Thus, in this set of JUnit-based tests we follow this good practice and “reset” the state of the test to an appropriate known value prior to each test being carried out. That way we get a much clearer picture of the “quality” of each component of the developed code.

You could illustrate this by introducing a logic error into the `withdrawMoney()` method and rerunning the tests. You will set that the “middle” test fails but that the others pass. This would not have shown up correctly if the `testDepostMoney()` method had not reset the “state” but had been written presuming a balance based on the previous withdrawal test.

- Introduce an error into the `withdrawMoney()` method and re-run the JUnit tests to satisfy yourself of this.

Also since we have decided to reset the state of each test to the same “opening” balance (i.e. £500 in each case) could you “refactor” (i.e. optimise) the test code by “abstracting out” the creation of the initial balance of the `BankAccount` class instance into the `setup()` method rather than including this separately in each of the three tests. Hint: to do this you would need to make your `testAccount` class instance variable global to the whole test class and create the new instance in the `setup()` method.

- Try to refactor the test class `BankAccountTest` to make use of the `setup()` method to implement this.

<u>**Week 3: Further JUnit Exercises and an Introduction to using CVS**</u>

The lab exercises in this third week are aimed at consolidating and building some further understanding in the use of the JUnit test framework. It will also provide a very brief introduction to the 5th objective of the practical work of this module. Thus the lab exercises this week are aimed at enabling you to develop an understanding of the following objectives of the lab work:

1. Understanding the use of CVS within the NetBeans IDE to access a central code repository
2. (beginning your) Understanding and using the core set of JUnit framework facilities to develop test programs for given software.

<u>Ex. 1: Using CVS within NetBeans to access code from a central repository</u>

<u>**Important Note:** if you are doing this exercise **at home** with any version of Netbeans from Version 7.1 or later, then you will have to install the CVS plug-in explicitly before starting these exercises by choosing **Tools -> Plugins** from the main menu of Netbeans and then installing the **CVS** plugin from the tab with the list of **available plugins**</u>

It was indicated in the first lecture in the Practical Lecture series that the use of a central code repository was essential within a modern team-based software development environment. In this central repository would be held the source code components making up the complete build. Holding them in a central repository makes the process of working on the different code components easier. Essentially, when a developer (or team) need to perform some work on a software component (which is either fully or partially completed), that code component is “checked out” from the repository and “downloaded” into the local (IDE) workspace of the developer’s machine. Once the developer has made changes on that component, then ultimately that component can be “checked in” to the central repository. That way there is always a single up-to-date repository of source code components.

Central code repository systems also provide <u>**version control**</u> for the purposes of <u>**Software Configuration Management**</u>. Whilst the “checking in” of the new component will “overwrite” the old version of the component, the repository has a system for maintaining the old versions, so that the code “history” is maintained and, if necessary, can be recovered as required by the development team. Whilst this module is not a software development module, it is important to give you some idea of the way in which such a system integrates with the tools used in a modern software development environment. Thus we will use the tool known as CVS (Concurrent Versions System) to provide a central repository of the source code for the lab exercises in this week’s lab.

The JUnit aspect of the lab exercises will involve being given information about the specification of the functionality of pre-prepared java classes. You will be required to develop test cases for these classes, write and execute them using the JUnit framework through NetBeans. However, the first example will also serve as an introduction to the use of CVS within NetBeans.

First Exercise:

This exercise will use a project which has a class based version of the problem (introduced in Tutorial 1) of finding the largest of three numbers. This project also has the same error in the algorithm as that from the tutorial (i.e. if z is the largest of the three values, but x is less than y, it doesn't find it).

The project has one class called `FindLargestOfThree`. This class has a single method with the following specification:

compute

```
public int compute (int x, int y, int z)
```

returns the largest of three integer number

Parameters:

`x, y, z` – the 3 integer numbers to be checked

Returns:

the value of the largest of the three numbers

The above form of specification of a class method is the "standard" Javadocs format which you often see in reference material.

The project for this class has already been prepared and has been placed in a CVS based central repository on a commercial Cloud-based CVS web-based hosting service for version control (Codesion/CloudForge). Thus your first task will be to "check out" this project from the central repository. The project is called **Week3LabExercise1** and it is stored (along with all of the other projects for this week's exercises in a folder in the CVS repository called **week3labs**. Whilst CVS has a command line interface, it is integrated into a variety of modern IDEs (including NetBeans)

Checking Out a Project from a CVS Repository

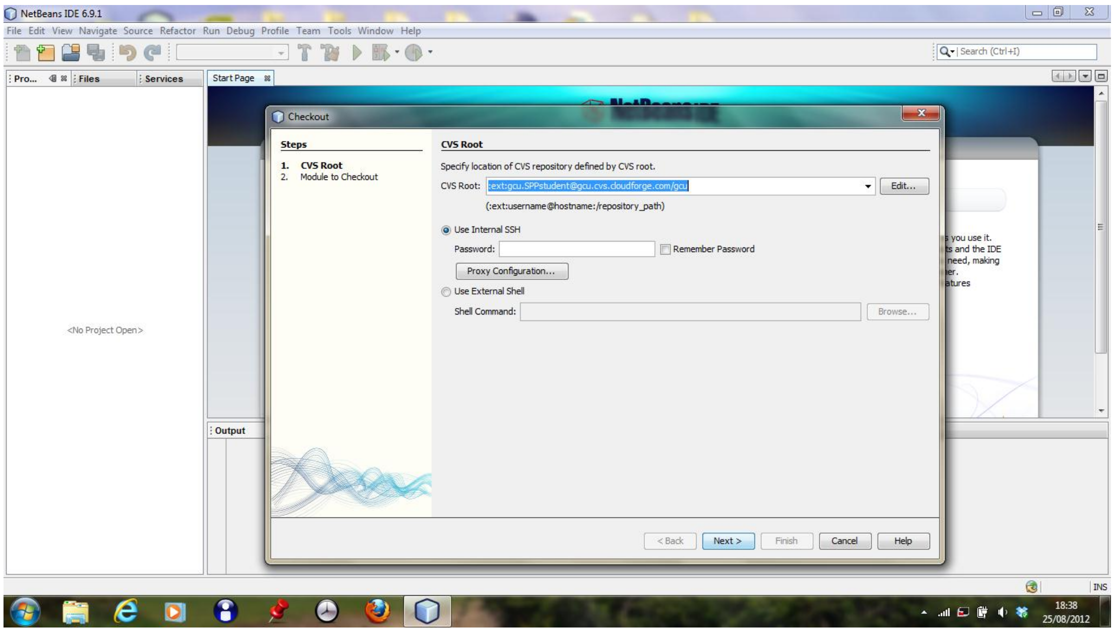
Continued OVERLEAF

Checking Out a Project from a CVS Repository

To check out the project from the CVS repository into Netbeans, do the following:

- Launch NetBeans
- From the NetBeans Menu, select **Team** -> **CVS** -> **Checkout …**

You will get the CVS Checkout wizard dialogue box (below):


Checkout

Steps

1. **CVS Root**
2. Module to Checkout

CVS Root

Specify location of CVS repository defined by CVS root.

CVS Root: :ext:gcu.SPPstudent@gcu.cvs.cloudforge.com/gcu

(:ext:username@hostname:/repository_path)

[● Use Internal SSH]

Password:

☐ Remember Password

○ Use External Shell

Shell Command:

- In the CVS Root box enter:
  :ext:gcu.SPPstudent@gcu.cvs.cloudforge.com/gcu [7]

(SPPstudent is the read only username for the gcu repository which has been set up)

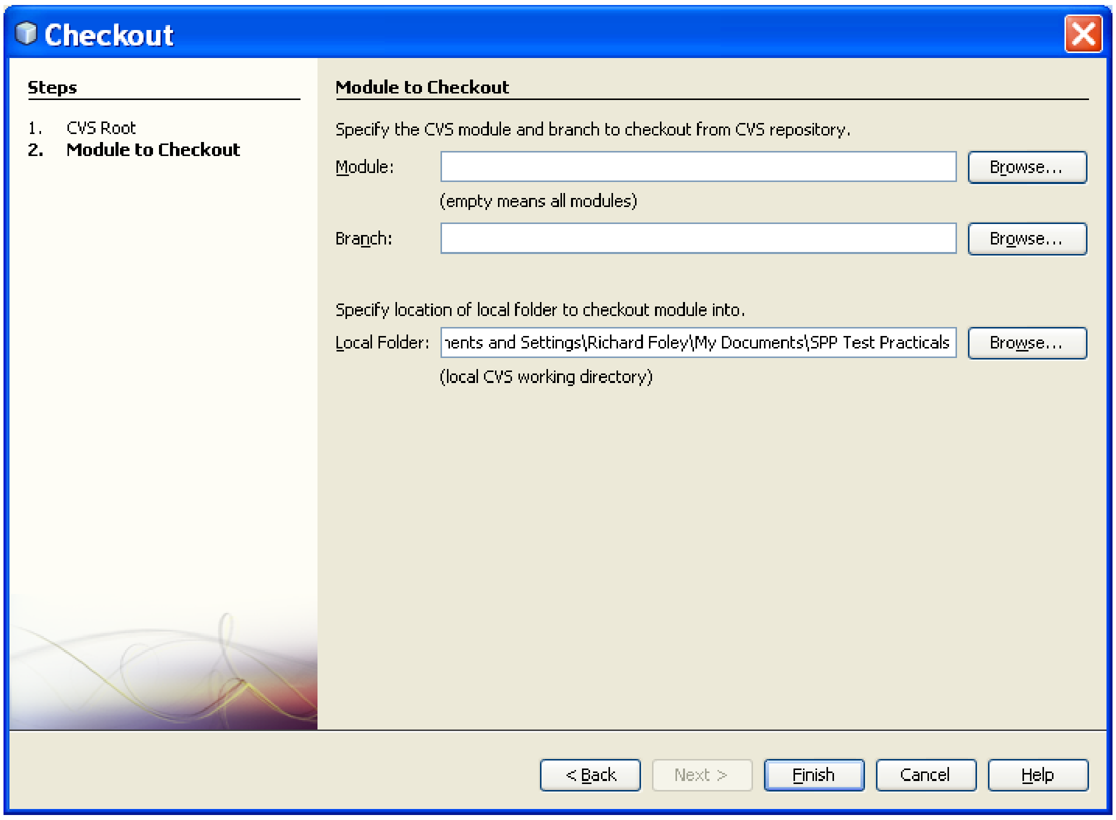
- In the password box enter:
  SPPstudent

(this is the password for this user name)

- Then click **Next >**

You should then see a progress bar appear at the bottom of the dialogue as NetBeans tries to connect to this web service. If you are successful the dialogue box will move to step **2. Module to Checkout** (over page). If this has not occurred the most likely reason is incorrectly entering the CVS Root username/password combination, thus check this carefully and try again.

[7] If you are attempting this from your home PC, you may have to use the alternate user name **:pserver:gcu.SPPstudent@gcu.cvs.cloudforge.com/gcu** as you may find that the :ext version of the user name does not allow final access to the repository contents due to network configurations.



Specify the CVS module and branch to checkout from CVS repository.

Module:

(empty means all modules)

Branch:

Specify location of local folder to checkout module into.

Local Folder: nents and Settings\Richard Foley\My Documents\SPP Test Practicals

(local CVS working directory)

In the **Module:** you will need to browse for the folder **week3labs** and then select the module (in this case the project) to checkout from the repository. (i.e. **Week3LabExercise1**). To do this you will need to expand the **/gcu** top level folder which you get in the dialogue which appears once you press the **Browse...** button to get the list of available projects. Note that this "root" (i.e. /gcu) folder will have several other folders as well not related to your module.

- Browse for the module name **Week3LabExercise1** (which is under the **week3labs** folder in the **/gcu** root repository) using the **Browse...** button and when you complete this task its pathname should be in the **Module:** box
- Under the **Local Folder:** box enter/browse for the folder in your local PC where you wish to place this checked out module. Please **note down the name of the folder** you intend to place the checked out project in. (As you do these exercises you will note that all of the Netbeans projects you checkout will be under a folder **week3labs** which is created on the first checkout.)
- Then click the **Finish** button

Again you will see a progress bar (this time in the bottom status bar of NetBeans itself) and if you are successful CVS will have placed this project in the specified folder and NetBeans will show a dialogue box asking if you wish to open the project. You accept this and open the project. If you were unsuccessful, then again check your entries for typos/correct browsing and try again.

Presuming that you have completed a successful checkout and opened the project, you should note (normally once you have started to make changes to the project), that the project icon (and various files) in the project pane will have a little blue "cylinder" icon beside it. This indicates that the project is "under version control". When a CVS-initialised project is created, there are additional CVS metadata folders/files included in the file structure which are used to maintain the versioning information required by CVS to keep track of the changes etc. being made. As you make any changes to any of the files when you are working on the project then you will see more of these appear as different changes are being made to the various files within the NetBeans project directory structure.

You will also, as you make changes see that the colours green, blue (and red) are used in the IDE to highlight changes to files and lines. Netbeans keeps track of the changes made from the initially checked out copy.

Green: indicated additions/new lines have been added
Blue: indicates changes made since the earlier (checked out) version
Red: indicates deletions have been made

<u>Testing the Class FindLargestOfThree using JUnit</u>

- Expand the project **Week3LabExercise1** in the project pane of **NetBeans**. You should note that the project only has two folders at present: for Source Packages and Libraries

Since the project on the repository does not, as yet, have any code for any test classes then no test folder will have been checked out. Thus your local project directory structure does not have a test folder within it and thus (for the moment) NetBeans does not recognise that there are any test packages/libraries available. Obviously in a (TDD-based) “production” development process, the current tests would be checked out along with the current production code. However, in this case you will have to manually add a empty test folder into the project folder’s structure. Thus:

- Select the **Files** tab for the project in NetBeans and fully expand the folder named **Week3LabExercise1**, so that you can see that there are only the **nbproject** and **src** folders.
- Create a new (empty) folder called **test** in this folder **Week3LabExercise1**, via right-click (on the folder Week3LabExercise1) **-> New Folder**.
- Return to the **NetBeans Projects** tab, you should now see that there are 4 folders under the project: i.e. Test Packages and Test Libraries should now have appeared.
- (If these additional folders have not appeared after a few moments then right click on the project name, and select Properties, the simply click OK on the Project properties dialogue box which appears. This will “force” NetBeans to resave the project data and the “missing” folders will “appear”.)

You should remind yourself of the specification of the module `compute()` and familiarise yourself with the source code of the `FindLargestOfThree` class. The `compute()` method has code very similar to the equivalent exercise in the week 1 tutorial. Again you may wish to compare this code with that tutorial handout.

You should now do the following (this should take some time and effort to complete):

- Consider, and note down 4 test cases which will **<u>fully</u>** test the method `compute()`

- Use the JUnit wizard (as you did in the week 2 Lab) in NetBeans to generate a test class skeleton for the `FindLargestOfThree` class
- Change the code inside the created `testCompute()` method to perform the four test cases you have selected.
- Execute these tests and view the results

If you "successfully" completed these tasks then 3 of your tests should have passed but one would have failed.

Correcting the code for the compute() method and re-running the tests

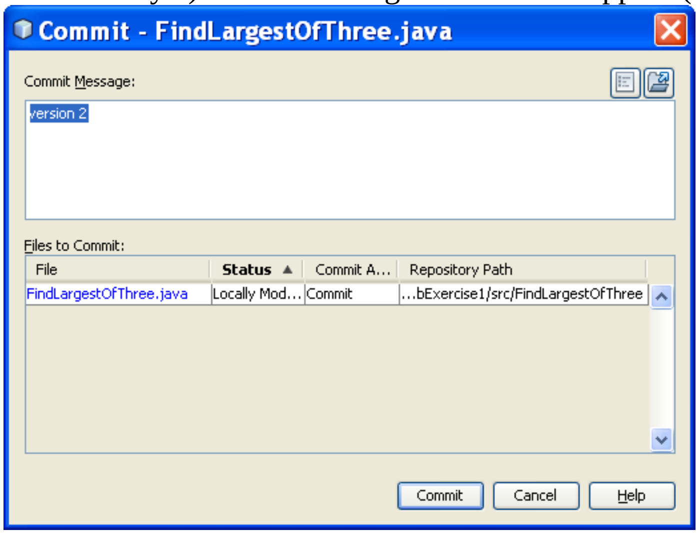
When you have successfully completed the previous task (i.e. have 3 passing tests and 1 failing test), you should now try to correct the code in the method `compute()` and re-run the tests. In this case it should be easy as you should simply follow the hint given in the comment near the end of the `compute()` method's code. As you do this you should see a green bar appear beside the line(s) you are adding, signifying local changes from the last checked out revision in the repository.

- Change the source code of the `compute()` method to correct the error and re-run the tests until you get all tests to pass

In a realistic team based software development environment (which is what we are trying to make you aware of here), the next step would be to "check in" or "commit" the changes in the code back to the repository. You will do this now via NetBeans, but just for demonstration purposes since when CVS within NetBeans trys to commit (and upload the new version of the class to the repository) it will fail at the last step since the SPPstudent username (since all students are "sharing it") has been set up with only read access to that repository! Still it should illustrate the process. Thus, once you have successfully corrected the `FindLargestOfThree`'s `compute()` method:

- rightclick on the FindLargestOfThree.java file in the NetBeans Project pane
- select **CVS** -> **Commit…**

The following (or similar style) Commit dialogue box should appear (as below):



version 2

File	Status	Commit A...	Repository Path
FindLargestOfThree.java	Locally Mod...	Commit	...bExercise1/src/FindLargestOfThree

- then put in a message to be stored as part of the versioning information, and click **Commit**

You should then see the bottom status bar indicating that the file is being uploaded back to the central repository. However, you will get (as I indicated) an error dialogue showing that it failed due to the SPPstudent account only having read access. However, if you have got this far you have successfully completed the first lab exercise for this week!

The remaining exercises follow:

Ex. 2: Further JUnit Exercises for week 3 lab

For each of the following classes:

- Develop a set of test cases, check out the corresponding source code in the CVS repository to an appropriate NetBeans Project, using the Create Test wizard (or by hand coding) implement a set of JUnit tests and test the code in each case. The examples here are not necessarily comprehensive classes, but are aimed at being sufficiently simple just to give you additional practice in using CVS and constructing JUnit tests within NetBeans.

Exercise a)

Palindrome.java
This class has one method:

isPalindrome
`public boolean isPalindrome(int num)`

Checks a number for a palindrome number

Parameters:
`num` – the number to be checked for a palindrome

Returns:
true if `num` is a palindrome number, false otherwise

Exercise b)

FindLowest.java
This class has one method:

findLowest
`public int findLowest(int[] values)`

Finds the lowest value in an array of integers

Parameters:
`values` – the integer array of values, which must be terminated by a dummy value -99

Returns:
the lowest value in the array of integers

Exercise c

SortList.java
This class has one method:

sort

```
public int[] sort(int[] values, int size)
```

sorts an array of integers into ascending order

Parameters:
`values` – the array containing the unsorted integer numbers
`size` – the number of elements in the array `values`

Returns:
a reference to the sorted array of values

Note: this example is more complicated to test that you may first think. When you call the method `sort` you must provide it with an array of a fixed size (you can make it any size you wish) and then after calling the method correctly, you must use `assertEquals` to check if each element of the returned array is now in the correct sorted order. You could do this by having separate `assertEquals` for each explicit array value. This, however, would be rather cumbersome the larger the array you used. Thus to do this you should use a `for` loop to cycle through each element of the returned array to check each element against another fixed array which contained the correctly ordered values.

Weeks 4-5: Further JUnit Exercises II – Testing in a Black Box Environment

The lab exercises in this 4-5th (and subsequent) week(s) are aimed at giving you further practice in the development of the use of the JUnit test framework.

Ex. 1: Testing code without reference to its source code

We have placed significant emphasis on the concept of TDD (i.e. Test Driven Development). As we should be aware, with TDD, tests should be written before any code is developed. Thus the tests should be written based only upon the specification of the code unit to be tested. Indeed this was one of the key reasons that a TDD approach produced better software, since it forces an early analysis of the specification of software.

As we shall see later in the Main Lecture Series, this strategy for Unit Testing is termed "Black Box" testing. With the strategy of Black Box testing, test cases are derived only through an analysis of the specification of the software (and with no consideration of the internal content or structure of the underlying code). It is for this reason, that such a strategy is also useful for high level (acceptance) testing, since it can be developed with the customer. After all the customer cannot (and isn't expected to) understand program code. Similarly, many organisations have testing (and analysis) groups separate from the development group. In many cases that group does not consist of people with the high level of software development experience and program coding skills as those of the development group. Thus a lot of software testing involves Software Engineering professionals, who are not "front line" coders and are not expected to have that same "front line" skill set!

As has also been emphasised, this module is NOT a software development module. However, as we can see from the preceding discussion, testing of code is not something which always requires a detailed understanding of the code (i.e. the developed software) and it is not uncommon for those not developing the code, and who are dedicated to other areas of the software lifecycle, to be involved in the testing process.

In TDD, the tests are written first (in JUnit) and then the development team write a minimal set of skeleton code which ensures that the tests can all be run and will all fail! The code is then developed and the tests re-run until all pass. Each time a testing set is executed; this gives immediate feedback about the "quality" of the code and which bits "work" and which bits "don't work". Once all tests pass, refactoring takes place, with the tests being re-run again to ensure that everything remains "green" and that the refactoring "Improves" the code, rather than "breaks" the code again.

Thus we will try to follow some of these principles as we go through the remaining exercises to be done with regard to the use of JUnit in this and subsequent labs. Indeed, from now on, (and again remembering that this is NOT a software development module) we will develop and execute all of our tests on software without even "seeing" the source code for the code we are testing!

To do this we will use **NetBeans** to create a skeleton (but "empty") project. Then rather than use the wizard to create tests based on the source code of a class, we will simply "load" a jar file with the executable JVM code for that class into the "empty" project, separately write a test class under the **Test Packages** folder and test the "byte code" version directly.

The first example of this method will be developing a test program to test a `BubbleSort` class. This class has one method with the following specification:

sort

`public int[] sort(int[] values)`

sorts an array of integers in ascending order (using the bubble sort algorithm)

Parameters:

`values` – the fixed size array containing the unsorted integer numbers

Returns:

a reference to the sorted array of values

Before creating the test program for this class, you must download, from GCU Learn the appropriate jar file(s) containing the "object" code for the `BubbleSort` class you are going to test. Thus do the following:

- Login to GCU Learn/Blackboard and go to the module:
  14/15 A - SOFTWARE PROCESSES AND PRACTICES
- Then from **Learning Resources -> Lab Work -> Week 4-5 Lab**
- Download the following two files from those in that folder:
  DummyBubbleSort.jar and
  BubbleSort.jar

A JAR file (or **J**ava **Ar**chive file) is a distribution version of the code of a java project. In this case the file **DummyBubbleSort.jar** contains a version of the `BubbleSort` class which is just a "dummy" and contains the skeleton, but no processing code. Whilst the **BubbleSort.jar** file contains the completed `BubbleSort` class.

The idea in this exercise is to create a program to test the `BubbleSort` class, i.e. write a JUnit test class with a simple test. Execute this using the **DummyBubbleSort.jar** version so that the test fails. Then re(run) the test on the **BubbleSort.jar** version so that the test passes.

Thus to create a program for this test, do the following:

- Launch NetBeans
- Create a new project called **BubbleSortTest** (accepting the initial defaults)
- Delete its **Main.java** class from its source package **bubblesorttest**
- Add to **test packages** a new package named **bubblesorttest**; and then
- Add to that test package a new java class named **BubbleSortTest(.java)**

In this class **BubbleSortTest.java** replace the created skeleton class `BubbleSortTest` with:

```
import bubblesort.BubbleSort;
import junit.framework.*;
public class BubbleSortTest extends TestCase {
   public void testSort () {
        BubbleSort testInstance = new BubbleSort();
        int values[] = {5, 2, 9, 4, 1, 3};
        int expected[] = {1,2,3,4,5,9};
        int[] actual = testInstance.sort(values);
        for (int i=0; i<values.length; i++){
             assertEquals("Testing sort", expected[i],actual[i]);
        }
   }
}
```

Note the following about this test class:

- It has an `import` statement, which imports the class `BubbleSort` from the package **bubblesort** (although we have not yet added the JAR file which contains this package and so the NetBean Editor should indicate the error that the package does not exist)
- It is a very simple version with no constructor, `setUp()` or `teardown()` methods
- It contains a simple test which creates an instance of the `BubbleSort` class, creates an array with unsorted values, creates another array containing the sorted sequence we expect, executes the `sort()` method and finally uses a `for` loop to test that each element of the actual output produced by the `sort()` method is equal to each corresponding element of the array contents/order expected.

To complete the first phase of this, we shall add the JAR file **DummyBubbleSort.jar** to this project. To do this, do the following:

- Right click on **Libraries** in the project pane; and then
- Select the option **Add JAR/Folder…**

From the resultant **Add JAR/Folder** dialogue box,

- Browse to the location of the file **DummyBubbleSort.jar;** and
- Select that file and click **Open** to add it to the libraries for this project

Return to the project pane and:

- Expand the **Libraries** icon to see that the JAR file **DummyBubbleSort.jar** has been imported into the project.
  (If the error, i.e. red squiggly underline on the `import` statement has not gone away, just double-click on the **BubbleSortTest.java** icon and that should refresh the compiler output to that window)
- Expand the JAR file and then the **bubblesort** package and you will see clearly what the JAR file contains.

- Execute the test, by right clicking the **BubbleSortTest.java** icon and selecting **Test File**

You should note that the code executes, but, of course, the test itself fails! That is because the version of BubbleSort in **DummyBubbleSort.jar** is simply a skeleton to allow you to confirm that your test(s) execute properly and fail.

Now what you want to do, is delete the **DummyBubbleSort.jar** from the project and load the **BubbleSort.jar** file into the project, re(run) the test, which should (hopefully) pass. To do this, do the following:

- Right click on the **DummyBubbleSort.jar** icon in **Libraries** and select **Remove** to delete that from the project
- Right click on **Libraries** in the project pane; and then
- Select the option **Add JAR/Folder…**
- Browse to the location of the file **BubbleSort.jar;** and
- Select that file and click **Open** to add it to the libraries for this project

- Then re-execute the test, by right clicking the **BubbleSortTest.java** icon and selecting **Test File**

This time the JUnit test output pane should be green indicating that the test has passed.

Ex.2: Further JUnit Exercises for week 4 lab

Exercise a)

Class `Factorial`, with one method

Compute:

```
public int compute (int number)
```

calculates the factorial of a given integer

Parameters:

`number` – the value for which the factorial must be calculated

Returns:

factorial of the value `number`, or -1 if `number` is negative

In the Week 4 Lab folder in GCU Learn, there are 2 JAR files related to this example are:

DummyFactorial.jar: where all aspects of the class should fail
Factorial.jar: the functionality should all be correct

Create a test program, called FactorialTest, write your tests, confirm they all fail when **DummyFactorial.jar** is used, and that they all pass when **Factorial.jar** is used.

When you complete this, consider the following and add more tests to the completed **Factorial.jar** test version as required:

- Two very simple tests (one with a single valid input and another with a single invalid input) is the most basic attempt, is that all you have done in your initial attempt?

- If so, do the following:
    - Amend your single valid input test to test several (i.e. at least 5) valid values. Try to use a `for` loop and two pre-initialised `array`s to do this rather than 5 separate asserts.
    - Add extra (and separate) valid and invalid tests to ensure that basic "boundaries" are tested.[8]

<u>Exercise b)</u>
Class `Fibonacci`, with one method

Compute:

`public int compute (int n)`

calculates a term in the fibonacci series

Parameters:

`n` – the term in the fibonacci series which is required

Returns:

fibonacci value of the `n`th term, or -1 if `n` is not greater than 0

The 2 JAR files related to this example are:

DummyFibonacci.jar: where all aspects of the class should fail
Fibonacci.jar: the functionality should all be correct

Create a test program, called FibonacciTest, write your tests, confirm they all fail when **DummyFibonacci.jar** is used, and that they all pass when **Fibonacci.jar** is used. Try to ensure that the set of tests you initial develop (and fail) are as full a set as possible, remembering the general principles from exercise a).

[8] Remember, that as well as negative values being invalid, the factorial of zero (0!) is **<u>valid</u>** and is the value 1.

<u>Exercise c)</u>

Class `StringPalindrome`

Method:

isPalindrome:

```
public boolean isPalindrome(String input)
```

Checks a string for a textual palindrome

Parameters:
`input` – the String to be checked

Returns:
True if `input` is a palindrome, false otherwise

In the Week 4 Lab folder in GCU Learn, there are 3 JAR files related to this example. These are:

DummyStringPalindrome.jar
IncompleteStringPalindrome.jar
StringPalindrome.jar

- With **DummyStringPalindrome.jar** all correct tests will fail. I.e. it incorrectly recognises non-palindromes as palindromes and vice versa. Thus a set of valid tests will all fail.
- With **StringPalindrome.jar** the class is fully correct.
- With **IncompleteStringPalindrome.jar** it isn't fully correct. Specifically it recognises some types of palindromes correctly, but not others.

By first writing a full set of tests and creating a test program called `StringPalindromeTest`, determine the error in the **IncompleteStringPalindrome.jar**

Weeks 6-7: Further JUnit Exercises III – Testing more complex objects

The lab exercises in this 6th week onwards are aimed at giving you further practice in the development of the use of the JUnit test framework.

You will find all of the **JAR** files needed for these exercises in GCU Learn under **Learning Resources -> Lab Work -> Week 6-7 Lab**

Exercise 1)
A class Prime is developed to manipulate a Prime Number[9]. It has the following specification: Class `Prime`, with:

Constructors:

`public Prime()`

creates a new `Prime` object without setting an initial value[10]

`public Prime(int p)`

creates a new `Prime` object whose value is set to the parameter `p`

Methods:

`public int getValue()`

returns the value of the number in the object

`public void setValue(int p)`

sets the value of the prime number in the object to the parameter's value

Parameter: p – the value to set the prime number

`public boolean isPrime()`

tests the value of the Prime object for a prime number value

Returns:

`true` if the object's value is a prime number, `false` otherwise

Hints and General Advice: you are expected to develop a **full** set of tests. Thus you should consider:

- Which categories of integer values can be prime numbers as well as which categories of numbers can definitely NOT be prime numbers.
- You should also adopt good practice and use separate test methods for distinct test behaviour.
- You should also test a series of typical values for both prime and non-prime numbers when you are testing "core" behaviour, rather than just one or two.
- You should also ensure that you test for "special cases".

[9] If you don't know the definition and characteristics of a Prime Number, look it up on Wikipedia. You might want to do this even if you think that you know what a Prime Number is as it gives a more detailed general specification and examples which may help you in developing a full set of test cases for the class.

[10] Do you know what java sets an `int` variable to is not initialised? If not either look it up or write a little of code to find out

- You should also use a meaningful name for each test method to help clearly identify which test method (and thus which test behaviour) failed, should a test fail.
- You should always use a text message in your assert and you should make it fully informative, by using appropriate String concatenations etc.
- As you develop your tests could you "extract" any "common" set up code into the `setUp()` method.
- You should also (this should really have been the first point to consider) note down on paper the details of each test case you intend to implement as a separate test method. This detail should include details of the test input you intend to use, the expected result and the purpose of the test.

For testing your JUnit tests in this exercise, two JAR files are supplied[11]:

> **DummyPrime.jar**: where both constructors just set the value to -99, `getValue()` always returns -99, `setValue()` has no effect, and `isPrime()` always returns `false`.[12]
>
> **Prime.jar**: with a fully correct version of `Prime`.

Exercise 2)

A class `Triangle` is developed which is to be used to represent a triangle. It has the following specification: Class `Triangle`, with:

Constructors:

```
public Triangle(int sideA, int sideB, int sideC)
```

creates a new `Triangle` object and sets its side lengths to the values of the three supplied parameters

```
public Triangle(Triangle t)
```

creates a new `Triangle` object whose contents are those from an existing `Triangle` object

Methods:

```
public int getSideA(),
public int getSideB(),
public int getSideC(),
```

These are "getter" methods which returns the value of the corresponding side of the associated triangle represented by the `Triangle` object

```
public boolean isTriangle()
```

returns true if the sides within the Triangle object can form a triangle, false otherwise[13]

[11] Remember that you will need to place an appropriate import statement in your Test Class since you are using the JAR file version rather than source code, i.e. `import prime.Prime;` in this example

[12] Not every exercise will supply both a "dummy" and fully functioning JAR versions

[13] Hint: three sides cannot form a triangle if any one of the sides is greater in length the sum of the other two sides. For all other properties of a triangle you should look up the term in Wikipedia!

`public boolean isEquilateral()`[14]

returns `true` if the Triangle represents an Equilateral triangle, `false` otherwise

`public boolean isIsosceles()`

returns `true` if the Triangle represents an Isosceles triangle, `false` otherwise[15]

`public boolean isRightAngled()`

returns `true` if the Triangle represents an right-angled triangle, `false` otherwise

For testing your JUnit tests, you will find one JAR file, **Triangle.jar** which contains a fully functional version of the `Triangle` class.

Exercise 3)

A drawing package uses a `Rectangle` object. It is used to represent a rectangle shape which is drawn on the visible screen. The drawing package presumes a screen with a ratio of 4:3 and a visible screen size of 800x600 co-ordinate values. It has the following specification: Class `Rectangle,` with:

Constructors:

`public Rectangle(int topX, int topY, int width, int height)`

creates a new `Rectangle` object

Parameters:

`topX, topY` – the x,y screen co-ordinate of the top left corner of the rectangle[16]

`width, height` – the width and height of the rectangle

`public Rectangle(Rectangle r)`

creates a new `Rectangle` object whose contents are those from an existing `Rectangle` object

Methods:

```
public int getTopX(),public int getTopY(),
public int getBottomX(),public int getBottomY()
```

These are "getter" methods which can be used to return the co-ordinate values for the top left and bottom right co-ordinates of the Rectangle object of the corresponding side of the associated rectangle represented by the `Rectangle` object

[14] Again if you are unsure of the definitions of these various triangles then look them up on Wikipedia at http://en.wiktionary.org/wiki/triangle

[15] This class presumes the "modern" rather than the "original" definition of an Isosceles triangle, see Wiktionary's definition of an Isosceles triangle at http://en.wiktionary.org/wiki/isosceles_triangle

[16] The screen co-ordinates system on a PC presumes that 0,0 is the top left corner of the screen and positive values for x and y are to the right and downwards respectively

`public int getWidth(),public int getHeight()`

These are "getter" methods which can be used to return the width and height of the rectangle

`public void rescale(double scaleFactor)`

This grows or shrinks the rectangle's size, keeping its top corner position fixed. If the `scaleFactor` is zero or negative the rectangle's size is not changed

Parameter:

`scaleFactor` – the amount by which the rectangle should be rescaled. The rescaled co-ordinates are rounded to the nearest whole value

`public void move(int moveX, int moveY)throws Exception`

this moves the Rectangle object by the x and y amount specified

Parameters:

`moveX, moveY` – the amount the rectangle's co-ordinates must be moved

Throws:

An Exception if the `Rectangle` would be completely outside the visible area of the screen where it to be moved. If this occurs the Rectangle's position is not moved.

I remind you that for this, and all exercises, you must consider the specification of the Class and develop your test cases before attempting any code for them in JUnit, you should note down your test cases in a table of the following form:

Class being tested

Method being Tested	Brief Description of test including explanation of test data

This table can then be used by you to plan the development of the JUnit test class and associated code.

For testing your JUnit tests, two JAR files are supplied:

DummyRectangle.jar: where both constructors do not set any values, and `rescale()` and `move()` have no effect.

Rectangle.jar: with a fully correct version of `Rectangle`.

Exercise 4)

A Date object has been developed which can contain any date in either the 20th or 21st century. It has the following specification: Class `Date`, with

Constructors:

`public Date(int day, int month, int year)`

Creates a new date object

Parameters:

`day, month, year` – integer values representing the date components

`public Date(Date d)`

Creates a new data object whose contents are those from an existing `Date` object

Methods:

```
public int getDay(),public void setDay(int d),
public int getMonth(),public void setMonth(int m),
public int getYear(),public void setYear(int y)
```

These are "getter" and "setter" methods which returns/sets the value of the corresponding day, month or year value of the associated date represented by the `Date` object

`public boolean validateDate()`[17]

checks the contents of the Date object for a valid date combination

Returns:

`true` if the values of day, month and year of the object make a valid date combination, `false` otherwise

`public String getDayOfWeek() throws Exception`

calculates the day of the week that the date falls/fell on[18]

Returns:

the `String` representation of the day of the week upon which the date falls/fell as "Monday", "Tuesday", etc.

Throws:

an exception if the date contained in the `Date` object is not a valid date combination

For testing your JUnit tests, two JAR files are supplied:

DummyDate.jar: where both constructors just set day, month and year to 0. "getters" always return 0 and the "setters" have no effect. `validateDate()` always returns false. `getDayOfWeek()` always returns "Monday".

Date.jar: with a fully correct version of `Date`.

[17] When testing `validateDate()` and `getDayOfWeek()` you may use a 2 dimensional array as in previous exercises, or you ould try to utilise the `ArrayList` collection class to create a list of the different `Date` objects with the different values you wish to use as your test cases.

[18] To help you in your development of test data for this method, see the tool on the web link http://katzentier.de/_misc/perpetual_calendar.htm. It will give the day of the week for any given date

Weeks 8-9: Further JUnit Exercises IV – Using setUp() and TestSuites

The lab exercises in this 8-9th week are aimed at giving you further practice in the development of the facilities of the JUnit test framework. Specifically some examples of the use of `setUp()` to provide a common test fixture for a set of tests and the construction of TestSuites to allow tests for more than one class to be run automatically and also to enable the tailoring of a set of tests for the same class.

You will find all of the **JAR** files needed for these exercises in GCU Learn under **Learning Resources -> Lab Work -> Week 8-9 Lab.**

This lab makes use of the class `Employee` which we introduced in Tutorial 4. Let us assume that it is now a bit more fully developed and its current specification is as follows: Class `Employee`, with:

Constructor:

`public Employee(int empNo, String name, int salary)`

creates a new `Employee` object whose values are set to the supplied parameter values

Methods:

```
public int getEmployeeNo()
public String getName(), public void setName(String name)
public int getSalary(), public void setSalary(int salary);
```

These are "getter" and "setter" methods which returns/sets the value of the corresponding class elements

`public int calculateAnnualBonus()`

calculates the annual bonus according to the company's bonus structure, i.e.

If the annual salary is up to £10000, then the annual bonus amount is 10% of salary; Salary between £10001 and £50000, then the annual bonus is 7%; Salary above £50000, then the annual bonus is 5%

Returns:

the calculated annual bonus for that employee

`public void increaseSalaryPercent(double percentageIncrease)`

applies a percentage pay rise to the salary of an `Employee`

Parameter:

`percentageIncrease` – the % value by which the salary should be increased. The new salary value is rounded to the nearest whole number value

`public void increaseSalary(int amountIncrease)`

increases the salary of an Employee by a fixed amount

Parameter:

`amountIncrease` – the fixed amount by which the salary should be increased.

Exercise 1) – Using `setUp()` and separating testing across several `TestCase` classes

In this exercise we will be creating two separate `TestCase` classs. One will be called `EmployeeInitTest.java` and it will deal with only the Constructor and the setter methods. The second will be called `EmployeeSalaryTest.java` and it will only deal with the methods:

- `calculateAnnualBonus()`,
- `increaseSalaryPercent()`, and
- `increaseSalary()`

In GCU Learn under **Learning Resources -> Lab Work -> Week 8 Lab** there are two JAR files, **DummyEmployee.jar** and **Employee.jar**. Both are versions of the `Employee` class. In **DummyEmployee.jar** none of the methods have any effect on the `Employee` class, whilst **Employee.jar** is a fully functioning version.

Do the following:

1. Create a new NetBeans Java project called **EmployeeTest**,
2. Delete its **main.java** source file,
3. Add the JAR file **DummyEmployee.jar** to its **Libraries**,
4. Add a new **Test Package** called **employeetest**,
5. Download the file **EmployeeInitTest.java** (from the **Week 8 Lab** folder on GCU Learn) and use it to create a java class of that name and contents in the **Test Package employeetest**,
6. Study the code in this java class **EmployeeInitTest.java** carefully. It this study you should note the following:
   - It only has 3 test methods, dealing with the constructor and the two setter methods of the class,
   - It uses the `setUp()` method (which as you should recall is automatically executed before each test method) to set up a simple test fixture, consisting of a single `Employee` object with some test values,
   - The declaration of the object which `setUp()` (re)initialises is declared "globally" to the whole of the class. If its declaration where placed inside the `setUp()` method then it would not be available to the other methods in the class
7. Run this test file and confirm that the tests all fail,
8. Now **Remove** the **DummyEmployee.jar** file from **Libraries**, replace it with the file **Employee.jar**, and confirm that these tests all now pass.
9. When this is complete, **Remove** the **Employee.jar file** and replace it back with the **DummyEmployee.jar** file in readiness for the next set of tests which you are going to develop.

In the next set of exercises you are going to create a second new test class in this project. You are going to create a separate `TestCase` class in the **employeetest Test Package** called **EmployeeSalaryTest.java** and it will only deal with the three related methods:

- `calculateAnnualBonus()`,
- `increaseSalaryPercent()`, and

- `increaseSalary()`

In this new `TestCase` class you will require to set up a common test fixture for the set of tests. That common test fixture should be a list of `Employee` objects, with the following information/values:

1st Employee:	647	"Smith, John"	5000
2nd Employee:	789	"Bloggs, Fred"	30000
3rd Employee:	895	"Jones, Thadeus"	60000

You could use a standard `array` (of `Employee` objects) to store the created list of objects, or you may be better to use the `ArrayList`[19] collections class. If you do use the `ArrayList` collections class, remember that you will have to import it for use by the program or the Java compiler will not recognise it, e.g.

```
import java.util.ArrayList;
```

Thus all three methods will be tested using this common test fixture. For each test you are required to perform the following tests only:

- For testing of `calculateAnnualBonus()`, just use the test fixture values
- For testing of `increaseSalaryPercent()`, presume that you are going to increase everyone's salary by 5%,
- For testing of `increaseSalary()`, the first employee will have a rise of £300, the 2nd £2500, and the 3rd £6000.

Now do the following:

1. Create another new class in **Test Package employeetest** called **EmployeeSalaryTest.java** (thus there will now be two classes in that **Test Package**)
2. Implement test methods for each of the 3 methods: `calculateAnnualBonus()`, `increaseSalaryPercent()` and `increaseSalary()` as per the criteria give above for both the `setUp()` and the tests themselves.
3. Run these tests on the **DummyEmployee.jar** version of the `Employee` class to confirm that they all fail.
4. Now **Remove** the **DummyEmployee.jar** file from **Libraries**, replace it with the file **Employee.jar**, and confirm that these tests all now pass.
5. Add another (separate) test method to **EmployeeSalaryTest.java** to deal with the "Boundary" values for calculating the annual bonus.

[19] The ArrayList class was also introduced in the Tutorial 4 exercises, which you should already have completed and understood. You would be well advised to restudy that as you prepare/develop your test methods in the rest of this lab exercise.

Exercise 2) – Creating and running `TestSuites`

The result from exercise a) should be that you have two separate `TestCase` classes dealing with different aspects of the testing of the single `Employee` class. Thus in this case, we have to "manually" execute both `TestCase` classes separately to fully test this class. As we have indicated, it is not unusual to have tailored tests or to split up tests for a single class across several test classes. In such a case we need to use the concept of a `TestSuite` to construct "Higher" level class structures which will enable these "lower" level tests to be automatically executed as a single action. Similarly, with the concept of Continuous Integration, we require to constructs sets of TestSuites to enable a complete test of the whole build of the systems and also sub systems. This process of the development and execution of TestSuites will be illustrated through the following exercises.

With the completed NetBeans project from Exercise a), do the following:

1. Add a new java class to the **Test Package employeetest**, and name it **EmployeeTestSuite.java**,
2. Type in the following as its contents:

```
import junit.framework.*;
public class EmployeeTestSuite extends TestCase {

    public static Test suite(){
        TestSuite suite = new TestSuite();
        suite.addTest(new TestSuite(EmployeeInitTest.class));
        suite.addTest(new TestSuite(EmployeeSalaryTest.class));
        return suite;
    }

}
```

3. Test this suite by right clicking on the **EmployeeTestSuite.java** icon in the NetBeans **Projects** window (as normal) and selecting **Test File**

You should note that running the test suite class will execute its created suite which consists of all of the tests in both `TestCase` classes. By expanding the green passed icon in the JUnit output window you can see that the 6 individual tests have been executed. The key information explaining this construct was given in Practical Lecture 5 (pages 34-37 of that yellow Lecture Notes booklet) and you should restudy that as you do this exercise to confirm your understanding/help you.

A test suite can be created to only execute specific tests from a particular test case class. Thus in the next exercise we will create another test suite which is going to consist of:

- all of the tests in the **EmployeeInitTest.java** class, but
- only the 2 salary increase tests from the **EmployeeSalaryTest.java** class

As indicated in Practical Lecture 5, if we wish to be able to create/specify specific tests from an existing `TestCase` class, then we need to add a constructor to that

`TestCase` class so that tests can be added individually to other TestCases/Test Suites. Thus do the following:

1. Add the following constructor method to the existing **EmployeeSalaryTest.java** class:

```
public EmployeeSalaryTest (String testName){
        super(testName);
    }
```

2. Add a new java class to the **Test Package employeetest**, and name i**t LimitedEmployeeTestSuite.java**,
3. Type in the following as its contents:

```
import junit.framework.*;
public class LimitedEmployeeTestSuite extends TestCase {

    public static Test suite(){
        TestSuite suite = new TestSuite();
        suite.addTest(new TestSuite(EmployeeInitTest.class));
        suite.addTest(new EmployeeSalaryTest("testIncreaseSalaryPercent"));
        suite.addTest(new EmployeeSalaryTest("testIncreaseSalary"));
        return suite;
    }

}
```

4. Test this suite by right clicking on the **LimitedEmployeeTestSuite.java** icon and selecting **Test File**

This time you will notice that only 5 of the total tests have been executed in this suite, i.e. all 3 from the `EmployeeInitTest` class and the 2 salary increase test methods from the `EmployeeSalaryTest` class.

Now do the following:

1. Create a new test suite for this project called OtherEmployeeSuite.java which will consist only of the constructor tests from EmployeeInitTest.java and your calculate annual bonus test(s) from EmployeeSalaryTest.java.

Exercise 3)

This exercise will use a new class called `Department` which can be used to contain the `Employee`s within a Department. Its specification is as follows: Class `Department`, with:

Constructor:

`public Department(String deptName)`

creates a new `Department` object whose name is set to the supplied parameter

Methods:

`public String getDeptName()`

returns the name of the Department object `public void addEmployee(Employee e)`

adds a given `Employee` object to the department

`public int noOfEmployees()`

returns the number of employees in the department

`public Employee searchForEmployee (int empNo) throws Exception`

searches for a given employee in that department

Returns:

the `Employee` object for the given `empNo` value if that employee is in that department

Throws:

an Exception if no employee with that employee number exists within that department

`public boolean deleteEmployee(int empNo)`

deletes an Employee with that employee number from the Department

Returns:

`true` if the employee with that number was in that department, in which case that employee will be deleted, `false` if the employee with that number was not in that department

In GCU Learn under **Learning Resources -> Lab Work -> Week 8 Lab** there are two JAR files, **DummyDepartment.jar** and **Department.jar**. Both are versions of the `Department` class. In **DummyDepartment.jar** none of the methods have any effect on the `Department` class, and the search method always return as found with a "dummy" `Employee` whose number is **999** and name is **"zzz"**, and the `deleteEmployee()` method always returns `false`. The file **Department.jar** is a fully functioning version.

In this exercise you should use the existing NetBeans project you have been developing during this set of lab exercises (I.e. EmployeeTest). Thus do the following:

1. Add the file **DummyDepartment.jar** to the **Libraries** of the project **EmployeeTest**
2. Add a new test case class to the existing test files in the **Test Package employeetest** of the project, name it **DepartmentTest.java**
3. Develop a full set of tests in that file **DepartmentTest.java** which will sufficiently test the `Department` class. Hint: you should use a `setUp()` method and in it you should create a test fixture of a `Department` with a similar set of `Employee` information as you have previously used.

\(OVERLEAF)

4. After completion of the full test case class, run those tests by right clicking the **DepartmentTest.java** icon in the projects window and selecting **Test File**. This should confirm failure as appropriate (remembering that `deleteEmployee()` always returns `false` and `searchForEmployee()` will always return a "dummy" `Employee` whose number is **999** and name is **"zzz"**, and not throw an exception).
5. Now **Remove** the **DummyDepartment.jar** file from **Libraries**, replace it with the file **Department.jar**, and confirm that these tests all now pass.
6. Finally, create a new test class called **SystemTestSuite.java**, which will automatically run all the tests from both **DepartmentTest.java** and the two original `Employee` test classes

Weeks 10-11: Software Processes & Practices (M3G605258) - Mock Coursework Test

Date: *Weeks 10-11*
Duration 90mins

This test consists of 4 questions all of which you should attempt. The individual marks (out of a total of 100) for each is shown against each question.

Preparation Instructions:

This is a CLOSED-BOOK test. **You may, however, bring into the test a single sheet of A4 paper with any notes of your choice typed/written on both sides of that sheet**. Otherwise, you are not permitted to use any other notes, books or reference material during the test. This includes electronic based material. You should bring with you a pen drive which should only be used for the purposes of back up and collection of your question attempts during the test. The test will commence at 10 minutes past the hour and you will be instructed by the lab invigilator when you can start the test. Until then do not open this test paper.

Prior to the instruction to commence the test, however, you should do the following:

- Log on to your PC and start the Java/Netbeans VMware application image
- Using a web browser from the VMware image, Log on the GCU Learn/GCU Learn for this module and download the file **MockTestWeeks10-11.exe** from **Learning Resources -> Mock Test** onto the **VMware Desktop**. This file is a self extracting zip file.
- Run this file and unzip its contents to the **VMware Desktop**. It will unzip as a folder called **MockTest** in which will be the NetBeans Projects and JAR files required for this test.
- Prior to commencing this test, **Rename this top level folder** so that the folder name is your own name in a Surname first format, e.g. Smith John
- The test will involve you adding/amending test code on the projects in this folder structure
- As you complete your attempt at each question you are instructed/advised to back up this folder structure on a pen drive. Thus you should insert your pen drive into the USB slot of your host PC and use Windows Explorer with copy and paste to make an initial complete copy of this folder onto the root folder of that pen drive. You would be advised to keep the Windows Explorer application open for ease of use during the test. If you have not brought a pen-drive then you should use your Network Drive instead for this back up purpose.
- Launch NetBeans and await the instruction to commence the test.

During the test you will/may need to write some answers on the test script. Thus complete your personal details here (printing clearly):

Name: …………………………………..

Programme: Circle your BSc programme
Computing(Information Sys Development)
Computing(Web Systems Development)
Computer Games (Software Development)

At the end of the test you are required to tick as appropriate to indicate which questions you have attempted:

Q1 ☐ Q2 ☐ Q3 ☐ Q4 ☐

Question 1 (20%)

Within the folder structure for this test there is a Netbeans project called **TimeTest**. In its **Test Packages** it contains a skeleton JUnit test class called `TimeTest` which is designed to test a simple java class called `Time`. This java class `Time` has a single method called `convert24to12hour()`. The specification of this method is as follows:

`public String convert24to12hour (String t)`

takes a `String` which represents a 24hour time value and converts it into a `String` representing its equivalent 12 hour am or pm time

Parameter:

`t` – a `String` which represents a 24hour time value, e.g. “0650”, “1245”, “2359”

Returns:

the equivalent 12 hour am or pm time `String` value in the form “NN:NNpm” or “NN:NNam” if the parameter `t` represents a valid 24hour time value, otherwise the `null String` is returned. In the returned `String` any leading zero is omitted, e.g. “0650” would result in “6:50am”.

Included in the root directory of the folder structure for the test are two jar files:

- **DummyTime.jar** is a version of the class `Time` which will always return the string value “dummy” no matter what input is given,
- **Time.jar** is a fully functioning version of the class `Time`.

You are required to fully develop the JUnit class `TimeTest` to execute the following specific tests only:

Input value	Expected Output
"0000"	"12:00am"
"1200"	"12:00pm"
"0300"	"3:00am"
"0650"	"6:50am"
"1059"	"10:59am"
"2159"	"9:59pm"
"2359"	"11:59pm"
"12309"	null
"650"	null
"2450"	null
"1360"	null
"0675"	null
"x650"	null
"1Q50"	null
"16Z0"	null
"165K"	null

As you develop your tests you may wish to use the **DummyTime.jar** version to first confirm all your tests fail, then replace this with the **Time.jar** version to confirm all your tests pass. However, you are not required to follow this pattern if you do not wish to. Either way, when you have completed your attempt copy the complete Test Folder structure from your (VMware) Desktop to your pen/network drive, overwriting the previous version on your drive.

Question 2 (30%)

Before you commence this question did you remember to make a copy of the test folder structure onto your pen/network drive after completing your attempt at question 1? You should do this after attempting each question (or more regularly) to ensure you do not lose any important work.

The folder **USTelephoneNoTest** is a NetBeans Project with a class `USTelephoneNoTest` which is intended to test a class `USTelephoneNo`. The class `USTelephoneNo` is a class designed to represent a telephone number from the United States of America. This class has only one constructor and two methods, as follows:

Constructor:

`public USTelephoneNo (String phoneNo)`

> this creates a `USTelephoneNo` object and sets its value to the supplied string (which represents the telephone number)

Methods:

`public String getUSTelephoneNo()`

> this returns the string held in the object representing the US telephone number

`public String validateUSTelephoneNo()`

> this checks the string representation of the telephone number held in the object and if it is a validly formed US telephone number, it returns the charge tariff for that type of number (either "standard" or "toll-free"). If the telephone number held in the object is not a valid string representation, then it returns the `String` value "invalid". A US telephone number has the following valid string format: "xnn-nnn-nnnn", x must be a numeric digit in the range [2..9] and n can be [0..9]. The first 3 digit block is for the area code of the telephone number. A "toll-free" charged number is one with the area code values of 800, 888, 877, or 866. All other valid numbers are "standard" charged numbers.

The NetBeans project called **USTelephoneNoTest** in the folder structure is a skeleton project to test this class. The jar file **DummyUSTelephone.jar** is a version where the constructor methods does not set any values and the `validateUSTelephoneNo()` method always returns the `String` "dummy" no matter whether the string is valid or not. The jar file **USTelephone.jar** is a fully functioning version.

You are required to fully develop the test class `USTelephoneTest` to test this class.

If you do not manage to fully complete this class/question then you should note down in the space below the series of separate classes of tests which you intended to develop for the whole class. This will help me give some credit if you do not fully complete the test code. Thus in that case, enter below brief details of each (equivalence) class of test you intending to carry out:

\(OVERLEAF)

Enter below brief details of each (equivalence) class of test you intended to carry out, if you were unable to fully complete this question:

When you have completed your attempt copy the complete Test Folder structure from your (VMware) Desktop to your pen/network drive, overwriting the previous version on your pen/network drive.

Question 3 (40%)

An airline has a loyalty card system whereby a customer collects points for flights which s/he takes. The airline has 2 different classes of travel, Economy Class, and Business Class. The number of points which a customer receives for a flight is based on the distance (in miles) for the flight and the class of travel booked. As the customer accrues flight points s/he is given a membership level. Under 20000 points is "Bronze", from 20000 to under 40000 is "Silver, and 40000 and above is "Gold". The java class `FlyMember`, specification given below, is used to model this situation:

Constructor:

`public FlyMember(int memberID, String name)`

creates a new `FlyMember` object with the given name and membership number. The number of flight points in a newly created object is set to zero. (You can presume that no validation is required on either of the constructor's parameters.)

Methods:

`public String getName (), public int getMemberID();`

returns the name and membership number respectively for the `FlyMember` object

```
public void setFlightMiles(int flightMiles)
public int getFlightMiles()
```

the "setter" method will set the total recorded number of flight points for a customer to the value of `flightMiles`, the "getter" method returns the total flight points for the customer

```
public boolean recordFlight(char travelClass, int
flightMiles)
```

adds the appropriate number of flight points to the `FlyMember` object based on the class of travel. Business class flights get 2*`flightMiles`. The characters 'E' and 'B' are used to denote the corresponding class of travel. The `flightMiles` is the designated distance of the flight. No flight distance in the airline is greater than 7000 miles. The method returns `true` if the supplied parameters represent a valid flight, `false` otherwise.

`public String checkMembershipLevel()`

returns the current membership level for the `FlyMember` object, either "Bronze", "Silver" or "Gold" *(Note: first letter of the string is a capital letter)*

The Netbeans project called **FlyMemberTest** in the folder structure is a skeleton project to test this class. There are two related jar files for this project. **DummyFlyMember.jar** is a version where the constructor and setter methods do not set any values. The `recordFlight()` method always returns `true` but does not update any flight points and the method `checkMembershipLevel()` always returns the `null` string The jar file **FlyMember.jar** is a fully functioning version.

You are required to fully develop the test class `FlyMemberTest` to test this class. When you have completed your attempt, copy the complete Test Folder structure from your (VMware) Desktop to your pen/network drive, overwriting the previous version on your pen/network drive.

If you do not manage to fully complete this class/question then you should note down in the space below the series of separate classes of tests which you intended to develop for the whole class. This will help me give some credit if you do not fully complete the test code. Thus in that case, enter below brief details of each (equivalence) class of test you intending to carry out:

When you have completed your attempt copy the complete Test Folder structure from your (VMware) Desktop to your pen/network drive, overwriting the previous version on your pen/network drive.

Question 4 (10%)

The folder **Courses** is a Netbeans Project with a class called: `Student`. This is part of an integrated student record system for a university which is currently being developed. This class is to contain brief student details and the marks out of 100% for each of 3 modules undertaken in a semester.

`Student`: this class is used to store the matriculation number, name, and the marks of the three undertaken by the student in the semester. It has a single constructor which sets the name and matriculation number values when a `Student` object is created. It also currently has the methods:

`public String getStudentName()`
returns the name of the student

`public int getMatricNo()`
returns the matriculation number of student

`public boolean addModuleMark(int mark, int moduleNo)`
records the mark for the given module number for a specific module for that student.

Parameters:
`mark` – the percentage mark 0-100 obtained for that module by the student
`moduleNo` – the module (either 1, 2 or 3) which this is the mark for

Returns:
`true` if the supplied parameters represents a valid mark and module no, `false` otherwise

`public int getModuleMark(int moduleNo)`
returns the mark for the given `moduleNo` (1, 2 or 3), if the value of `moduleNo` is not a valid module the value -2 is returned

`public int getNoOfModuleMarksEntered()`
returns the number of modules for which marks have been legitimately added

`public boolean makeDecision() throw Exception`
this processes the marks for the 3 modules recorded for the student and determines if the student has passed or failed the semester. The pass mark for each module is 40% and a pass is given only if all 3 modules are above the pass mark. This method with throw an Exception if a full set of marks has not been entered for the student.

`public int calculateAverage() throw Exception`
this calculates and returns the overall average mark (rounded to nearest integer) across the modules for the student. This method with throw an Exception if a full set of marks has not been entered for the student.

The NetBeans project has both source code and test class code for this class. The test classes are in development and there are two test classes (`StudentInitTest` and `StudentProcTest`) covering the set of tests for this single production code class. The class `StudentProcTest` is not fully complete. Thus you are required to do the following:

Separately execute the test classes: `StudentInitTest` and `StudentProcTest`. You should note that all of the `StudentInitTest`s pass, but both of the tests in `StudentProcTest` fail (i.e. `testMakeDecision()` and `testCalculateAverage()`). These two tests fail because the code for the `setUp()` method in the test class has not been written yet.

i) Develop two test suites for this project, adding the appropriate Test class for each, and any other code as required, to the project:

The first is to be a Test class called `UniversitySuite`, which will run all of the tests in both test classes `StudentInitTest` and `StudentProcTest` as a single test suite.

The second, a Test class called `ModuleMarksTestSuite`, this is to run only the following:

- the test method: `testValidModulemarks()` from the class `StudentInitTest` and
- both test methods from the class `StudentProcTest`

ii) For the test class `StudentProcTest`, now write code in the `setUp()` method which will initialise the globally declared `ArrayList` of `Student` objects (named `students`) with the following details for the following 4 students:

Matric No	Name	Mark	Mark	Mark
20091231	“xxxx, xxxx”	60	60	60
20091232	“xxxx, xxxx”	23	70	55
20091233	“xxxx, xxxx”	60	25	65
20091234	“xxxx, xxxx”	50	60	28

If you complete this correctly and run the test class `StudentProcTest` then its tests should pass, as should all of the tests in the `UniversitySuite.`

When you have completed your attempt copy the complete Test Folder structure from your (VMware) Desktop to your pen/network drive, overwriting the previous version on your pen/network drive.

END OF TEST

<u>Instructions to students for the end of the test:</u>

\(GIVEN OVERLEAF ON THE NEXT PAGE)

Instructions to students for the end of the test:[20]

When the lab invigilator instructs you to stop at the end of the test you must close your NetBeans application.

You should then make your final copy of the complete Test Folder structure from your (VMware) Desktop to your pen/network drive, overwriting the previous version on your pen drive. When you have done this you should **then make a zip file version** of that folder (on your pen drive) as follows:

- **Right-click** on the folder and select the **Send to** option on the menu,
- Select **Compressed (zipped) Folder** from the subsequent menu

This will create a zip file with the same name as your folder. Check that this had been done. However, do not close your VMware application or Log out from your PC yet!

You should then submit this zip file containing your attempted solutions via the Assignments mechanism of GCU Learn as follows:

- Login to the Software Processes and Practices GCU Learn module,
- Select **Assignments** from the left hand pane to go to the Assignment page
- Select the assignment link **MOCK CLASS TEST – Submission Test Assignment** from the subsequent Assignment page,

You should have the page: **Preview Upload Assignment: MOCK CLASS TEST - Submission Test Assignment**

- Scroll down this page until you see the **Browse My Computer** button of the option Attach File,
- Press the **Browse My Computer** button, navigate to and select your zip file,

You should see your selected file loaded when you return back to the Upload Assignment page. To complete the submission:

- Scroll further down the page and press the **Submit** button to get the **Review Submission History** page, which should confirm the details of the uploaded file you have submitted, then
- Press the (pale yellow) **OK** button

When you return to the Assignments page you can select the MOCK CLASS TEST – Submission Test Assignment link again to check that you have submitted your file correctly.

Once you have completed this, DO NOT yet log off from your PC! Go to the lab instructor and ask him/her to confirm that he has received your submission. If there is a problem the instructor will ask for your pen drive to make a manual copy of your submission.

Once that is done log off from your PC although: **All students must remain in the lab until instructed to leave by the lab invigilator.**

[20] Obviously your attempt at the Mock Test does not need to be formally submitted. However, you should follow these instructions so that you can test out the GCU Learn/Blackboard based Assignment submission process for yourself before the real test!

Appendix 1: Lab Work - Completion Checklist

Student Name: …………………………………………………………..

Programme: …………………………………………………………

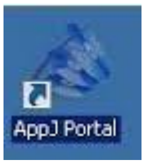
Lab	Done (Tick)	Date Completed
Pre-Objective:		
• Downloading Netbeans/Java for Home Use		
Week 1:		
• Exercise 1		
• Exercise 2		
Week 2:		
• Exercise 1		
• Exercise 2		
• Exercise 3		
• Exercise 4		
Week 3:		
• Exercise 1		
• Exercise 2a		
• Exercise 2b		
• Exercise 2c		
Weeks 4-5:		
• Exercise 1		
• Exercise 2a		
• Exercise 2b		
• Exercise 2c		
Weeks 6-7:		
• Exercise 1		
• Exercise 2		
• Exercise 3		
• Exercise 4		
Weeks 8-9:		
• Exercise 1		
• Exercise 2		
• Exercise 3		
Weeks 10-11 – Mock Coursework Test:		
• Question 1		
• Question 2		
• Question 3		
• Question 4		

Appendix 2: Using NetBeans 7 via Application Jukebox from the standard University PC Desktop

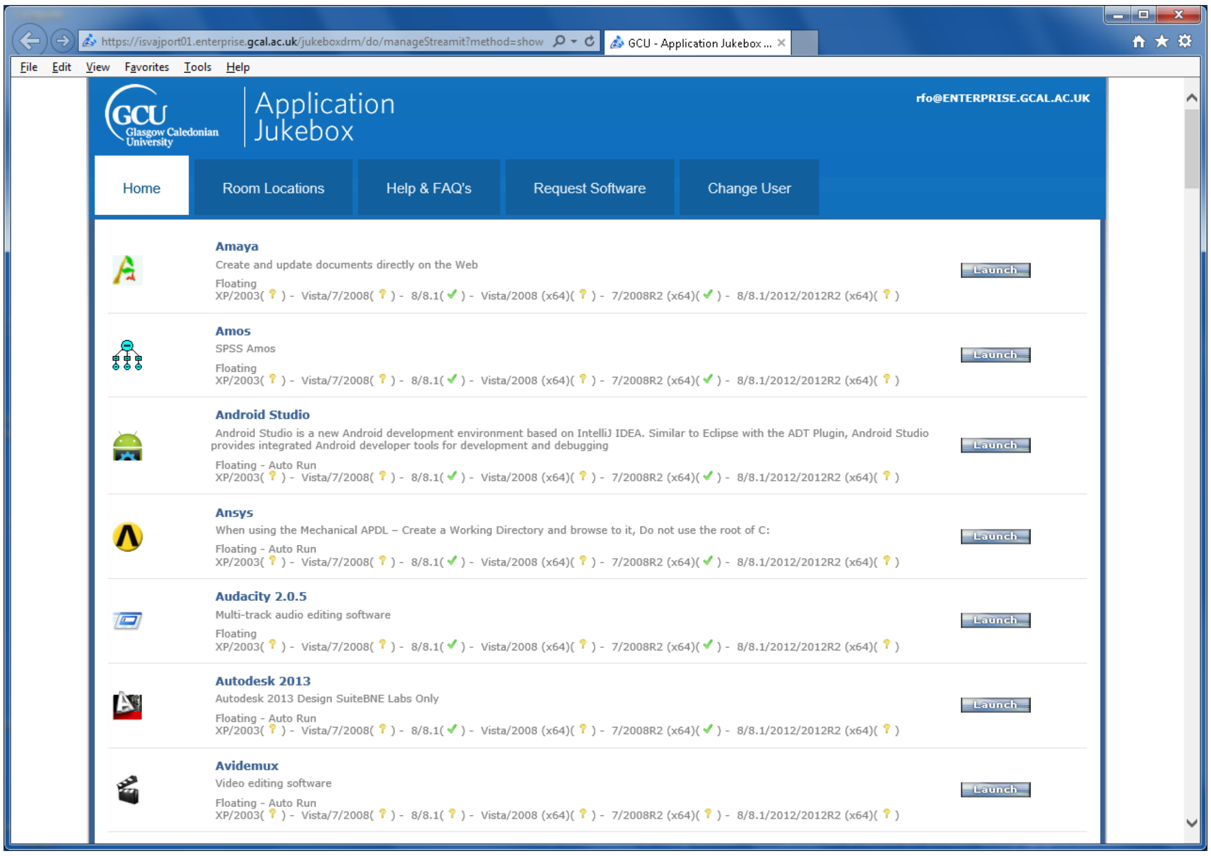
Whilst you are expected to use the Windows 8 (VMware) virtual machine with Netbeans 8.0, these are only available on specific (mainly) George Moore Building (GMB) labs. However, the University also has a system, known as **Application Jukebox**, which is deployed on the standard PC Desktop on all University PCs. Thus you make wish to use this at times if you are unable to access a GMB lab with the VMware image and wish to do some independent study of the lab material.

The version deployed via **Application Jukebox** is Netbeans 7, but that is quite sufficient for our purposes and the interface is very similar. To launch this Netbeans version via Application Jukebox, do the following:

- Launch the Application Jukebox portal via the shortcut on the desktop. The shortcut should look like:



Please note, however, that there may be 2 similar looking shortcuts on the desktop. The one you should launch is named **Application Jukebox Portal**. This shortcut launches Internet Explorer with an alphabetical list of the available applications. This should look as follows:


https://isvajport01.enterprise.gcal.ac.uk/jukeboxdrm/do/manageStreamit?method=show
GCU - Application Jukebox ...
File Edit View Favorites Tools Help
GCU Glasgow Caledonian University
Application Jukebox
rfo@ENTERPRISE.GCAL.AC.UK
Home
Room Locations
Help & FAQ's
Request Software
Change User

Amaya
Create and update documents directly on the Web
Floating
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(✓) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

Amos
SPSS Amos
Floating
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(✓) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

Android Studio
Android Studio is a new Android development environment based on IntelliJ IDEA. Similar to Eclipse with the ADT Plugin, Android Studio provides integrated Android developer tools for development and debugging
Floating - Auto Run
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(✓) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

Ansys
When using the Mechanical APDL – Create a Working Directory and browse to it, Do not use the root of C:
Floating - Auto Run
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(✓) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

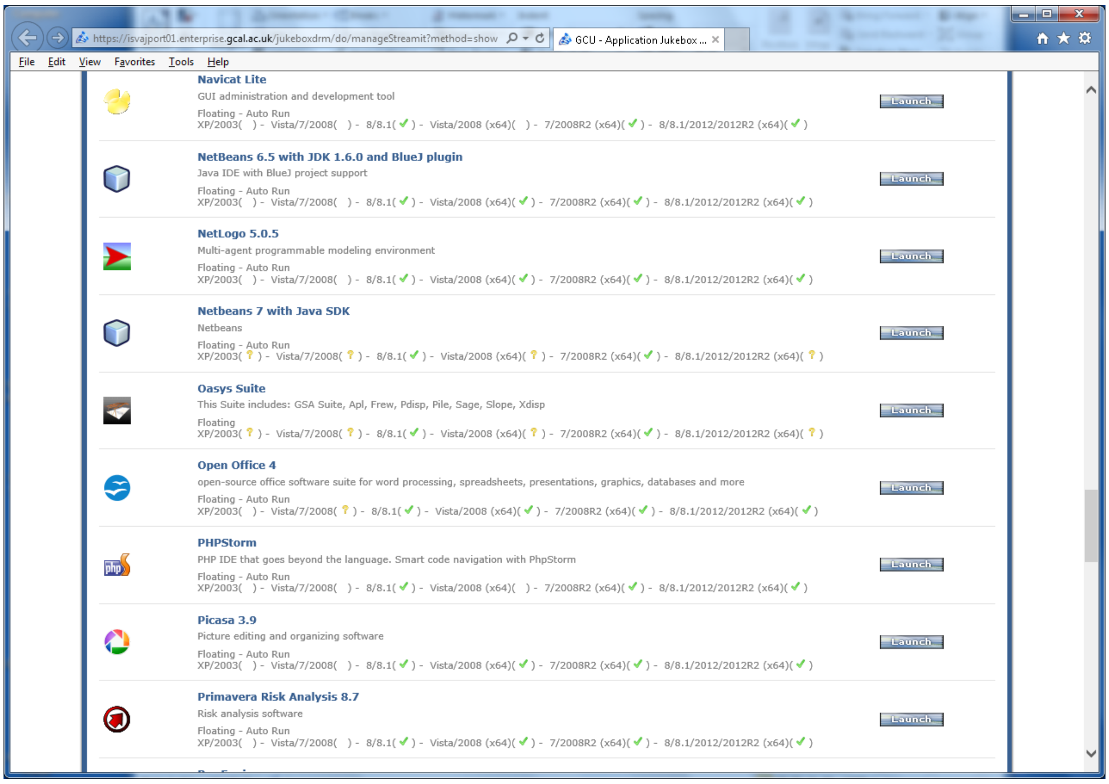
Audacity 2.0.5
Multi-track audio editing software
Floating
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(✓) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

Autodesk 2013
Autodesk 2013 Design SuiteBNE Labs Only
Floating - Auto Run
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(✓) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

Avidemux
Video editing software
Floating - Auto Run
XP/2003(🔑) - Vista/7/2008(🔑) - 8/8.1(🔑) - Vista/2008 (x64)(🔑) - 7/2008R2 (x64)(🔑) - 8/8.1/2012/2012R2 (x64)(🔑)
Launch

The list of available packages via Application Jukebox is lengthy and in alphabetic order. Thus:

- Scroll down until you see the application entitled **NetBeans 7 with Java SDK** (Note that you have the correct one, since there may be also a NetBeans 6.5, as per this screenshot)


https://isvajport01.enterprise.gcal.ac.uk/jukeboxdrm/do/manageStreamit?method=show
GCU - Application Jukebox ...
File Edit View Favorites Tools Help

Navicat Lite
GUI administration and development tool
Floating - Auto Run
XP/2003( ) - Vista/7/2008( ) - 8/8.1(✓) - Vista/2008 (x64)( ) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

NetBeans 6.5 with JDK 1.6.0 and BlueJ plugin
Java IDE with BlueJ project support
Floating - Auto Run
XP/2003( ) - Vista/7/2008( ) - 8/8.1(✓) - Vista/2008 (x64)(✓) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

NetLogo 5.0.5
Multi-agent programmable modeling environment
Floating - Auto Run
XP/2003( ) - Vista/7/2008( ) - 8/8.1(✓) - Vista/2008 (x64)(✓) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

Netbeans 7 with Java SDK
Netbeans
Floating - Auto Run
XP/2003(?) - Vista/7/2008(?) - 8/8.1(✓) - Vista/2008 (x64)(?) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(?)
Launch

Oasys Suite
This Suite includes: GSA Suite, Apl, Frew, Pdisp, Pile, Sage, Slope, Xdisp
Floating
XP/2003(?) - Vista/7/2008(?) - 8/8.1(✓) - Vista/2008 (x64)(?) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(?)
Launch

Open Office 4
open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more
Floating - Auto Run
XP/2003( ) - Vista/7/2008(?) - 8/8.1(✓) - Vista/2008 (x64)(✓) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

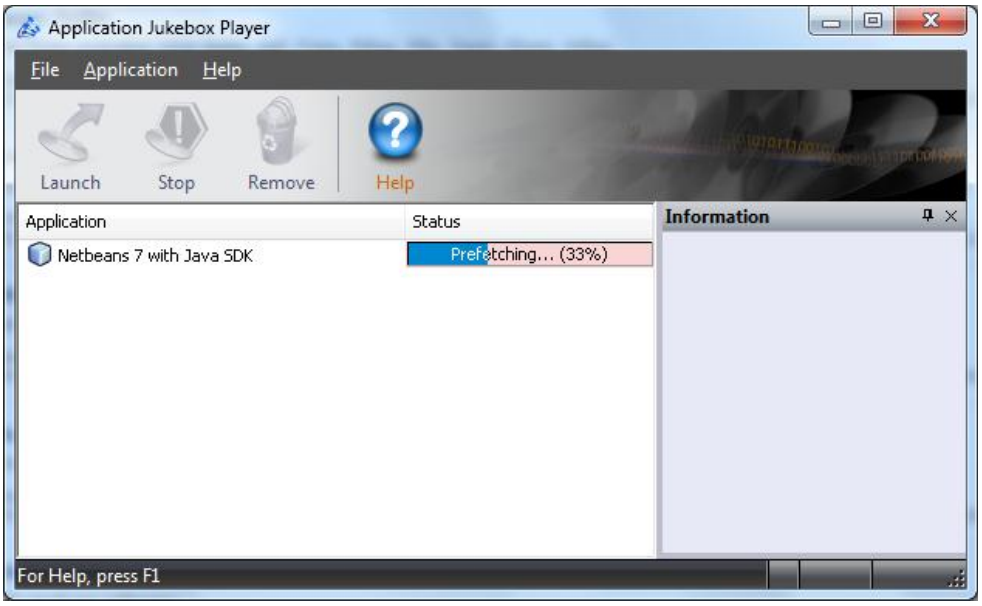
PHPStorm
PHP IDE that goes beyond the language. Smart code navigation with PhpStorm
Floating - Auto Run
XP/2003( ) - Vista/7/2008( ) - 8/8.1(✓) - Vista/2008 (x64)( ) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

Picasa 3.9
Picture editing and organizing software
Floating - Auto Run
XP/2003( ) - Vista/7/2008( ) - 8/8.1(✓) - Vista/2008 (x64)(✓) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

Primavera Risk Analysis 8.7
Risk analysis software
Floating - Auto Run
XP/2003( ) - Vista/7/2008( ) - 8/8.1(✓) - Vista/2008 (x64)(✓) - 7/2008R2 (x64)(✓) - 8/8.1/2012/2012R2 (x64)(✓)
Launch

Then:

- Launch **NetBeans 7 (with Java SDK)**, by pressing the (blue) **Launch** button beside the entry. This will then launch Application Jukebox (see screenshot below) with this application. After a few moments as it fetches and initialises, it will automatically start the application and you are ready to go!



NOTES

NOTES

NOTES

NOTES