COMP220/285 Lab sessions 1-3

Contents

General Notes

Please work through the lab assignments in sequential order, so don't start a later lab assignment until you have finished the earlier assignment. If you are not able to finish the all the tasks in the lab session, continue yourself in your own time. Notice within the lab work you will experience using Ant and JUnit as standalone tools as well as part of an integrated environment such as Eclipse an ability for work in both contexts (standalone and IDE) is an important software engineering skill. This is because the power and complexity available by writing your own standalone scripts may not always be available from the IDE.

Note, when running the example scripts, you can always copy and paste them from this document.

Getting started

Installation

First follow the instructions (video and text) on Canvas to download the zip file and set up Ant/Junit.

Remember you need to call setpaths, everytime you open up a new command prompt for the session.

Open a command prompt.

Task 1 Checking your ANT install

ant –version

You should see something like this (the actual version and build date may be different)

M:\ ant -version

Apache Ant(TM) version 1.10.3 compiled on March 24 2018

Task 2 Checking your JUnit install

As part of the set up for the labs... you need to add the following classpath, like so:

Run this from the command prompt:

set classpath=%classpath%;c:\java\junit4.12\hamcrest-core-1.3.jar;

To just run JUnit with no tests, type the following

java org.junit.runner.JUnitCore

Now to run a set of example tests

First copy this test class into your current directory

Download from Canvas, the Source Code Module the file ExampleTests.class

and the type the following:

java org.junit.runner.JUnitCore ExampleTests

You should usually get the following (sometimes you will get test errors, see source code to find out why), run the a few times to see what happens.

```
JUnit version 4.12
.....I.....I.....
Time: 0.02
OK (50 tests)
```

Have a look at the file ExampleTests.java (download code) and you can see the list of all the test cases. Each dot shows a test successfully executed. We can also see in the output the "I" this indicates tests that are ignored (not actually run)

Here is an example with a test that fails, not in this text there is an E in the output indicating a test has failed.

```
JUnit version 4.12
.....I.....E.............................I..........
Time: 0.03

There was 1 failure:
1) testf12(AllTests)
org.junit.ComparisonFailure: expected:<[]OK> but was:<[NOT]OK>
        at org.junit.Assert.assertEquals(Assert.java:115)
        at org.junit.Assert.assertEquals(Assert.java:144)
        at AllTests.testf12(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:483)
        at
org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(Framework
Method.java:50)
        at
org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCal
lable.java:12)
        at
org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMe
thod.java:47)
```

```
        at
org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMet
hod.java:17)
        at
org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
        at
org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRun
ner.java:78)
        at
org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRun
ner.java:57)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at
org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at
org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at
org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at
org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.junit.runners.Suite.runChild(Suite.java:128)
        at org.junit.runners.Suite.runChild(Suite.java:27)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at
org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at
org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at
org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at
org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.junit.runner.JUnitCore.run(JUnitCore.java:137)
        at org.junit.runner.JUnitCore.run(JUnitCore.java:115)
        at org.junit.runner.JUnitCore.runMain(JUnitCore.java:77)
        at org.junit.runner.JUnitCore.main(JUnitCore.java:36)

FAILURES!!!
Tests run: 50,  Failures: 1
```

Note some tests may work on some test runs and not on others, for example if a test fails due to the timeout of a request the failure to run this test may be dependent on the current state of the computer running the text.

Task 3 JUnit documentation review

The document for JUnit should be at

http://junit.org/junit4/javadoc/latest/

Start your favourite browser and open and book mark the file.

Task 4 Ant documentation

The Ant document is available at

https://ant.apache.org/manual/

Use the browser to review the documentation.

Task 5 Running the Eclipse IDE

To start Eclipse do the following

Start -> Java Apps -> Eclipse

Note the actual Eclipse directory is C:\JAVA\eclipse

The latest version of Eclipse will load, note that Eclipse is a large application and takes a while to load.

Selecting a workspace

A workspace is the running working area where all your projects are created, edited, saved and built. You may be asked to select a workspace, if so pick your M: drive and create a folder with a sensible name (e.g. M:\comp220\eclipse_projects). Note in the lab the workspace may have been pre-selected for you automatically as M:\eclipse.

Remember all the project work you are creating with Eclipse will be stored in this workspace directory, if you want to switch to a new workspace, you may do this by choosing File-> Workspace.

When you start Eclipse you will be presented with a screen as shown in Figure 1.

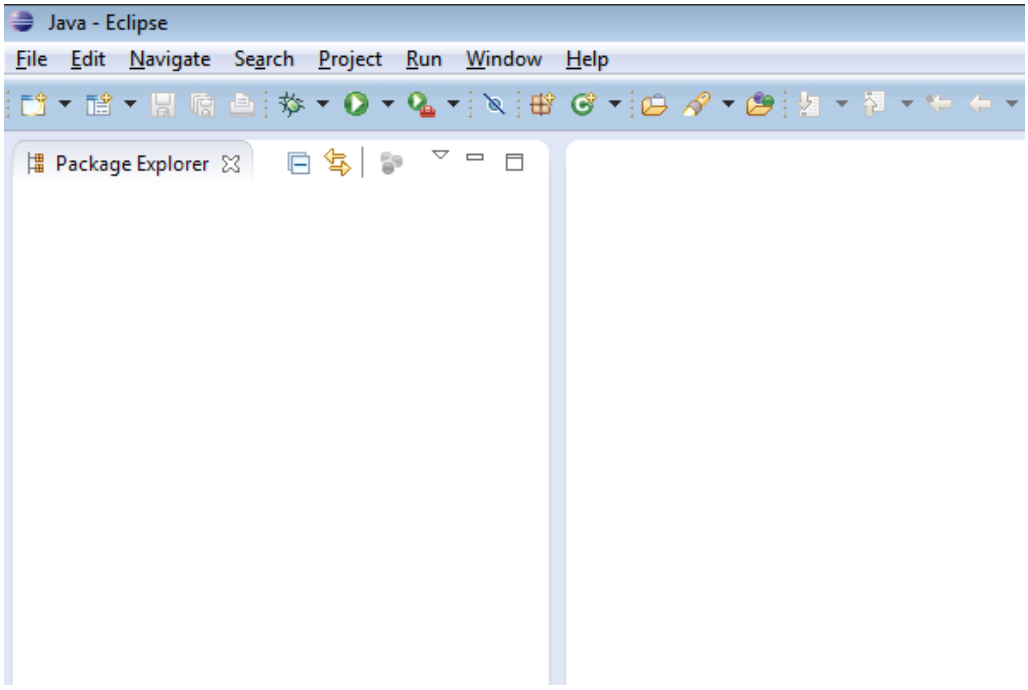

Figure 1 Eclipse start up screen

Starting with Eclipse

Perspectives

Likely as not, you will be presented with the Java perspective as shown in Figure 1, this shows you the menus and options suitable for Java™ development. You can switch perspective depending on the type of task you are involved with, for example you can change to the debug perspective by choosing Window-> Open Perspective-> Debug, try it and see how the display changes.

Choose the correct perspective dependent on your task is important as it makes it easier to see what is happening.

Task 6 Creating a new Eclipse project

Set the perspective back to Java. Choose File -> New -> Java Project and you should be presented with the window shown in Figure 2, choose a name for your project.. Example1 and then click Finish.

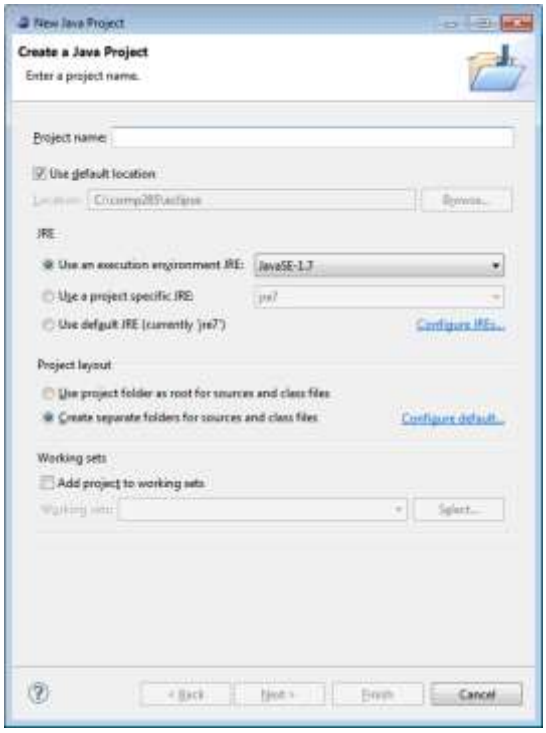

Figure 2 Creating a project

Now follow the steps outlined in the lectures, to create some new classes and build and execute your new project.

Task 7 Eclipse and JUnit

In this task you will add a set of tests to your Eclipse project.

Now follow the step by step instruction in the slides, trying to repeat (understand and remember) everything that was demonstrated in the lectures. In particular you should start with the example project in the lecture notes, add the tests first following the test-first philosophy.

Note for the lab test you may be asked to perform similar tasks and resolve similar issues, so practise these techniques so that you can complete the task quickly and accurately.

Task 8 Running a Simple Ant Build File

First create a directory to store your build.

mkdir buildexample1

Now add the following two files

Main.java:

```
public class Main {

    public static void main(String args[]) {
        for(int i=0;i<args.length;i++) {
            System.out.println(args[i]);
        }
    }
}
```

build.xml:

```
<?xml version="1.0"?>
<project name="firstbuild" default="compile" >
  <target name="compile">
    <javac srcdir="."
           includeAntRuntime="no"/>
                  <!-- It is usually best to set includeAntRuntime to
"false" or "no"
                       so the script's behaviour is not sensitive to the
environment
                       in which it is run. See
C:\JAVA\Ant1.8.1\docs\manual\index.html
                       on <javac> task.
                  -->
    <echo>compilation complete!</echo>
  </target>
</project>
```

Now run Ant from the current directory

ant

and look at the Ant output, including the contents of the current directory.

RUN Ant again. Do you see any difference? Try to understand what it means and why.

Run Ant again with the content of Main.java changed trivially (say, by adding an unnecessary space and saving). Again, what is the difference and how can it be explained?

What if <javac> task in build.xml is misspelled, say, as <javaac>? run it, and after realizing what happened recover build.xml back.

Run it with omitted the end tag </target> in build.xml and see what is the effect. Do not forget to recover build.xml back.

Run it, with misspelled srcdir attribute as sourcedir in build.xml. Do not forget to recover build.xml back.

If your build file has no errors, there may be errors because compiler fails to compile your code.

Delete the semicolon after println call in Main.java. Run Ant. In this case build.xml is not responsible for build failed. Do not forget to recover Main.java back again.

The key point: failure of a single task halts the entire build resulting in build failed. There is no point in packaging or delivering a project if it did not compile.

Looking at the build in more detail:

Try the command ant with verbose mode (twice—when Main.class does already exist, and when it does not exist yet (or just deleted))

ant -verbose

What is the difference? Pay attention to and explain these lines of the Ant output:

ant -verbose

...

compile:

[javac] Main.class skipped - don't know how to handle it

[javac] Main.java omitted as Main.class is up to date.

...

The above was quite a simple laboratory work. You should try to get a complete understanding. Later the level of difficulty will increase as we will consider more complex tasks.