Lab 1

Be sure that you have added the bin directory to the PATH variable (in the system variables). You can see whether this has already been done by typing in a command window

```
        java –version
```

This lab walks you through the steps of working with Java and Eclipse; you will not develop much code in this lab.

---

1. *Create your own Java application and run it.*

   a. Use a text editor to create a class Welcome that follows the same syntax as the Hello class discussed in the Lesson 1 slides. Write it so that the word "Welcome" is printed to the console when the application runs.

   b. Save your class Welcome as a file named Welcome.java. (You can pick any location on your computer to save it.)

   c. Open a command window and the current directory to the directory in which Welcome.java has been placed.

   d. At the command prompt, type

      ```
      javac Welcome.java
      ```

      The javac.exe compiler will silently compile without a message (unless you typed something wrong) and return you to the command prompt. However, you should see in the same directory as Welcome.java a new file called Welcome.class.

      (For more excitement, try compiling like this:

      ```
      javac –verbose Welcome.java
      ```

      In this case, the compiler will tell you many details of the compiling process.)

   e. Now at the command prompt type

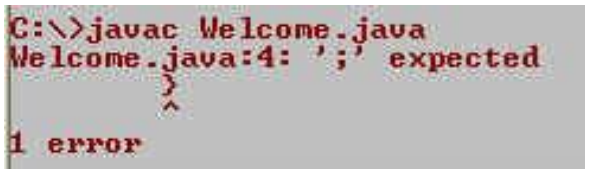
      ```
      java Welcome
      ```

      Unless you made a typing mistake, you should see output like the following:

      ```
      C:\>java Welcome
      Welcome
      ```

f. Now see what happens when you introduce a coding error. Go into Welcome.java and remove the semicolon from the line System.out.println("Welcome"). Then save.

g. Now try to run javac against the new file (as above). You will see something like the following:



```
C:\>javac Welcome.java
Welcome.java:4: ';' expected
        }
        ^
1 error
```

The compiler has indicated that it found one compiler error, and has attempted to show you where it occurs. In the source file, it tells you that the error has been noticed on line 4. Here is the source code:

```
class Welcome {
   public static void main(String[] args) {
      System.out.println("Welcome")
   }
}
```

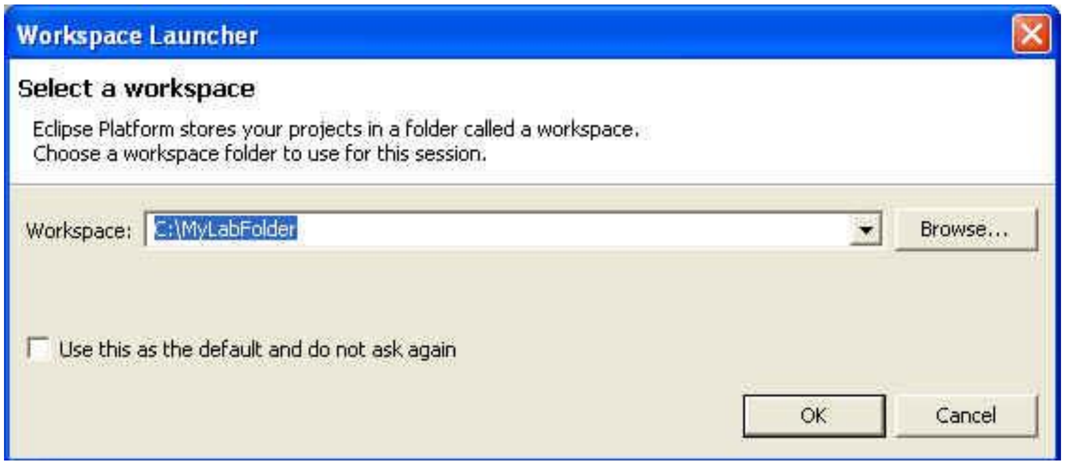
Line 4 contains the brace that immediately follows the missing semicolon. The message also alerts you that a semicolon is missing.

h. Now, if you want, insert the semicolon in the correct place, save, and compile again. The code should compile and run as before.

2. *Create a Java program in Eclipse.* You will create another simple program, this time using the Eclipse IDE.
    i. Eclipse will ask you to select a folder to use as a *workspace* folder. Within that folder, Eclipse will create one new sub-folder for each new *project* you create. Inside a project, you will create your java classes. Usually you will place your classes in one or more *packages* (also represented by folders).

        So, to start, decide where you want to place your workspace for this exercise.

    j. Start Eclipse. It will ask you to locate your workspace folder:



Select a workspace

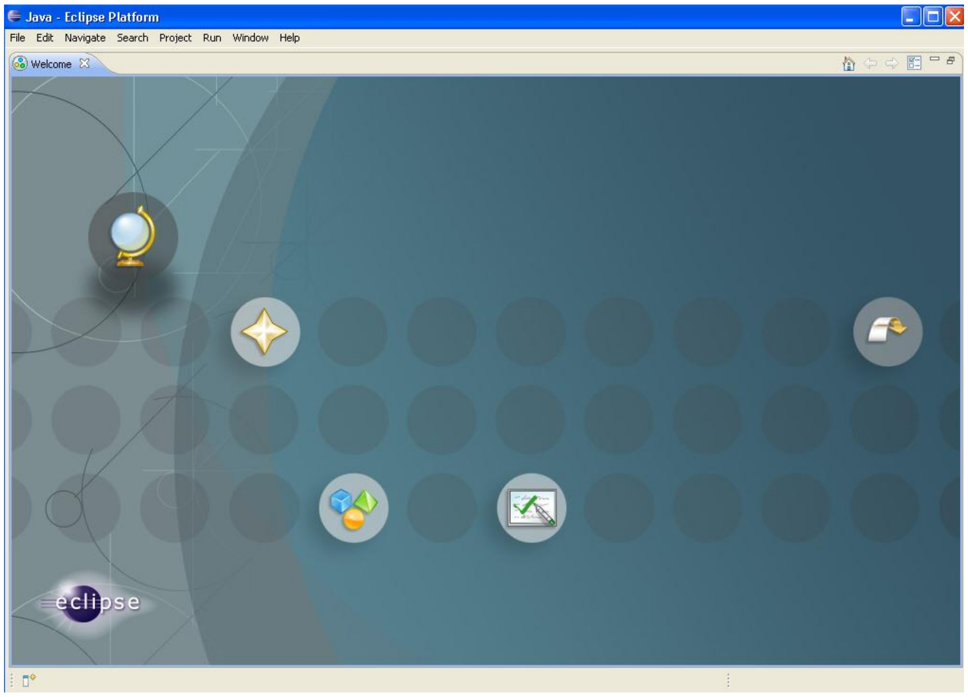
Eclipse Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\MyLabFolder

Use this as the default and do not ask again

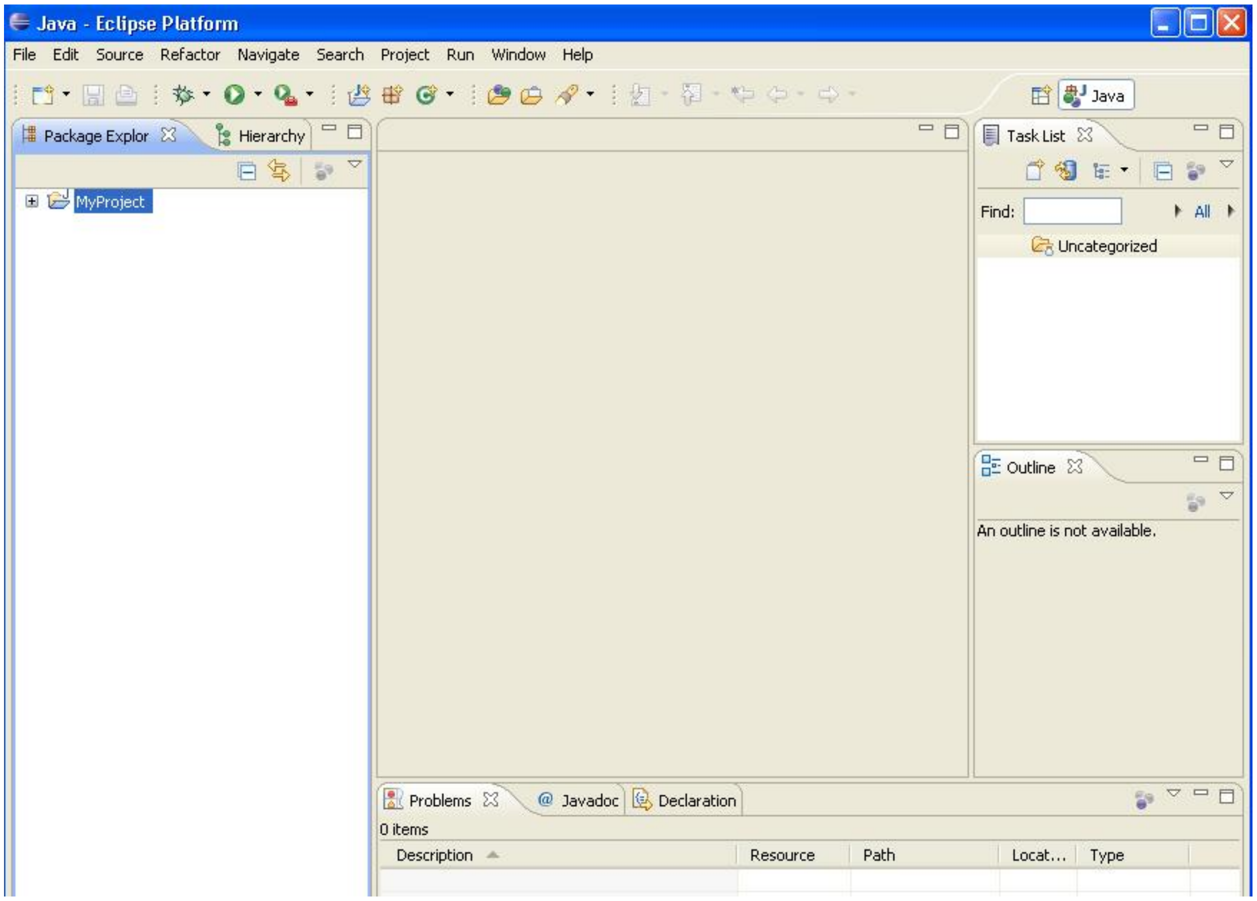
        Browse to locate the folder you have decided to use.

    k. If this folder has not been used as a workspace before, you will see a Welcome screen like this:



l. Close the Welcome tab. By default, you will see windows and tabs that are called collectively the "Java Perspective" – this is a view of files and other options that is convenient for Java development (other perspectives support other types of development).

m. Eclipse does not allow you to create Java classes without placing them in an existing project. So, you must first create a project. Right click on the leftmost panel, click New > Java Project. Give it the name MyProject, and click the Finish button at the bottom.



An outline is not available.

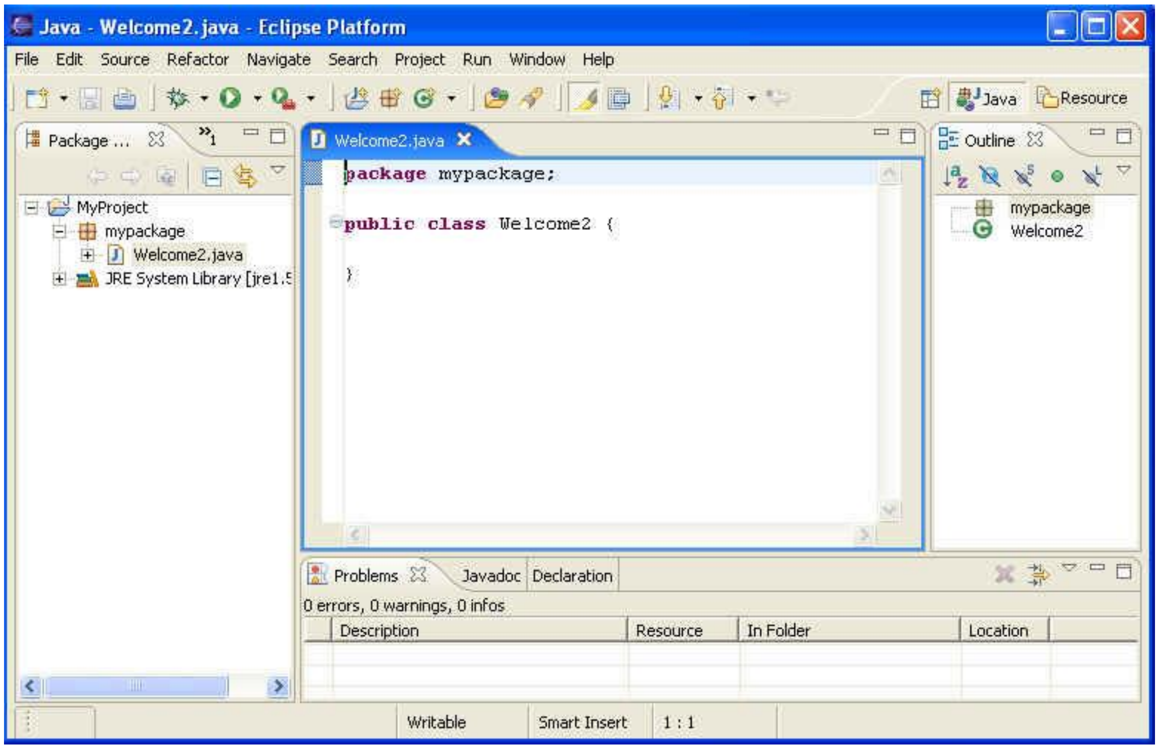
0 items

n. You can create your class inside this new project. It is better to place your Java classes inside *packages* (which help to keep your classes organized). So create a package by right clicking on MyProject in the leftmost panel, and selecting New > Package and give it the name `mypackage`, and click ok.

Naming Convention. Project names are always capitalized, but package names always are written without capitals and without underscores.

When you create a package, Eclipse automatically provides a src folder, indicating that your source code will be placed there. Your output files (after compiling) are placed by default in a bin directory, at the same level as src.

o. Finally create your Java class inside the package by right clicking on the package mypackage and selecting New > Class. Give the class the name Welcome2 and click Finish. Your new class file will show up in the editor.



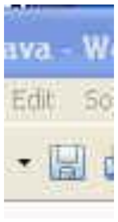
```
package mypackage;

public class Welcome2 {

}
```

0 errors, 0 warnings, 0 infos

p. The code you will write in this class will have two methods. One of the methods, like Welcome.java, will be main(String[] args) and the other will be named welcome(). Be sure to use the "static" keyword for both. The main method will call the welcome method.

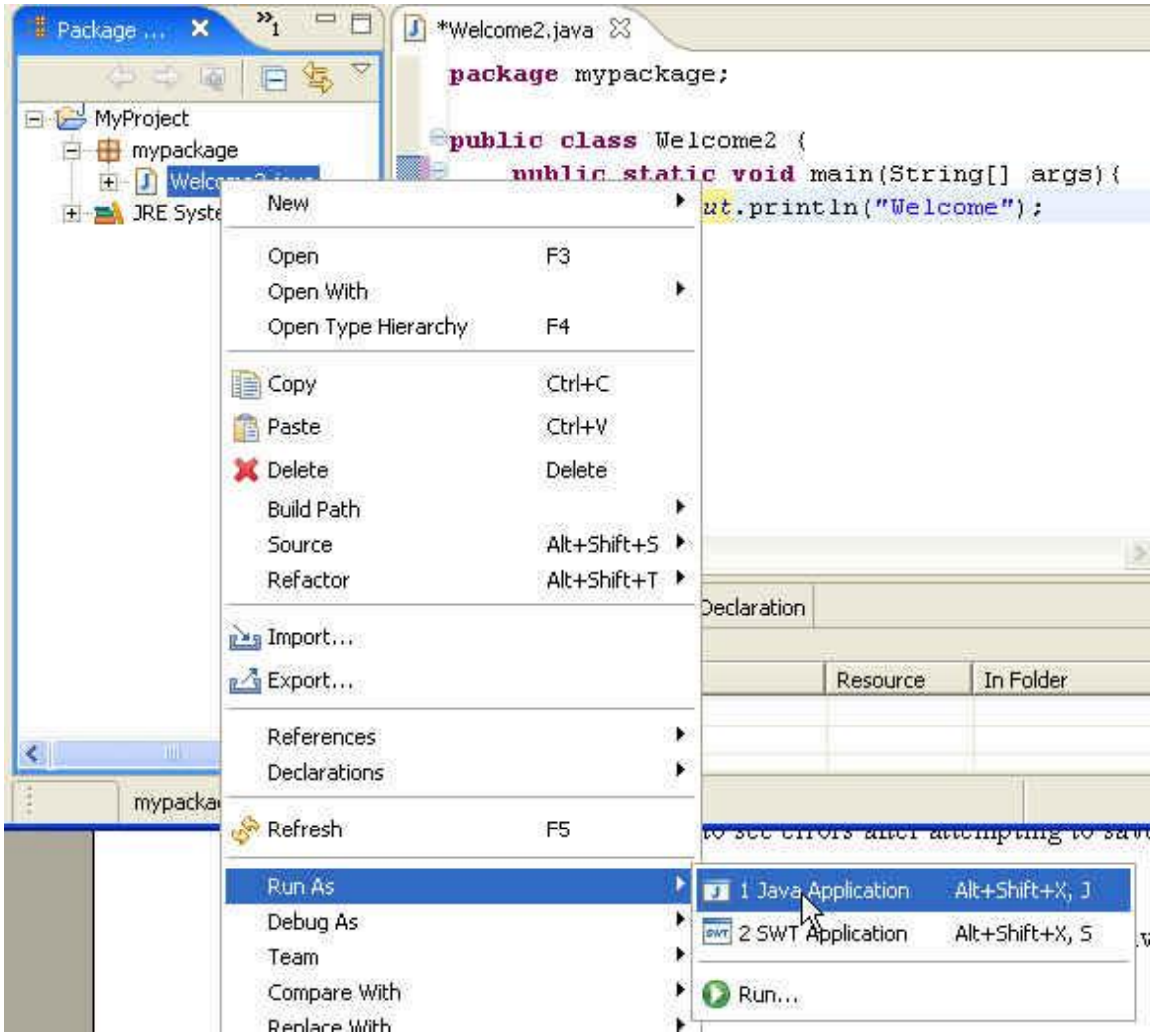
```
public static void main(String[] args) {
    welcome();
}
public static String welcome() {
    return "Welcome";
}
```

q. Save your work by clicking the icon



In Eclipse, when you save, the compiler is run automatically, so you can sometimes expect to see errors after attempting to save.

r. If you typed everything in correctly, no errors will have occurred. Run your program by right clicking on the filename Welcome2 in the left panel, then selecting Run As > Java Application:



```java
package mypackage;

public class Welcome2 {
    public static void main(String[] args){
        ...ut.println("Welcome");
```

k. In the console window at the bottom, you will see your output.

l. Now remove the semicolon from your file and save. Notice the error indicators on the file. Also, you can read the compiler error message by going to the Problems tab at the bottom panel.

m. Now reinsert the semicolon and Save.

m. Add a Test class: Create a new Java class TestWelcome and insert a method testWelcome(). Type the annotation @Test just above testWelcome(), as in the following:

```java
@Test
public void testWelcome() {
}
```

n. To fix the compiler error on @Test, click the red/yellow marker in the margin and select the option "Add JUnit 4 to your build path."

o. Write your test method inside TestWelcome2. It should look like this:

```
public void testWelcome() {
    String result = welcome();
    assertEquals("Welcome", result);
}
```

To fix the compiler error, click the red/yellow marker in the margin and select the option "Add static import for Assert.assertEquals".

p. Run your test by right clicking on TestWelcome2 class and opt for Run As JUnit test. You should see "green"!

q. Use the instructions given in the slides to deploy your Welcome2 application. You may deploy either with a batch file launcher or by creating a runnable jar file.

3.