Introduction to Eclipse

We will be using Eclipse in this course to create and run Java programs. Eclipse is a platform for application development. A separate handout posted to `Course Documents>Week 1` describes how to obtain and install Java and Eclipse. This handout will take you through the process of creating, compiling and running a Java program using Eclipse.

Eclipse is also installed in both CIS labs (Smith 212 and Smith 234) and you are welcome to work in either one (Java tutors are available in Smith 212). It is your responsibility to remove your files from the lab computer prior to leaving the lab.

1 Getting started with Eclipse

Specifying the Eclipse workspace

When you start Eclipse, you are asked to specify the directory in which to store your projects. By default, a directory called `workspace` within the directory in which Eclipse is installed is used. It's fine to use this directory on your own computer. However, when you are working in the lab, you should store your files to your M: drive [1]. **Don't forget to log out from the lab computer when you are done; otherwise someone else could get access to your personal M: drive.**

If you would like to switch the workspace to another location after starting Eclipse, go to `File>Switch Workspace` and select the desired folder.

Eclipse Perspectives and Views

The Eclipse workbench provides several *views* and *perspectives* to access information regarding a project. A *view* is a window that displays specific kinds of information regarding tasks. Each *perspective* is a collection of views designed for a certain kind of project. We are going to be using the `Java` perspective. Later in the course, we will also be using the `Debug` perspective.

Within the `Java` perspective, the following views will be used most often:

- `Navigator` - provides access to project files
- `Package` - shows java source files grouped by packages (i.e., folders)
- `Problems` - displays syntax and runtime errors
- `Console` - displays the textual input/output from running a Java program (the program interaction window)

[1] The M: drive is your personal space on the Bentley server; to access it, you must be logged on to the Bentley network. You (and only you) have access to your M: drive, so keeping your files there while working on a computer in a CIS lab guarantees that no one else will be able to access your code.

The `Java` perspective is the default. If it isn't already selected, you can select it by choosing `Window>Open Perspective>Other>Java`. Next, select the `Navigator` view by choosing `Window>Show View>Navigator` to see and access the contents of each project directory. The `Navigator` view is displayed on the left side of the Eclipse window. Select the `Console` view in the same manner and notice that it usually appears at the bottom of the window.

Creating a new project

To create and run a Java program in Eclipse you must form a *project* for it.

Create a new Java Project by selecting `File>New>Java Project`. Next, enter any project name you would like into the Project name field.

From the `Project Layout` box, **select the first option** – Use project folder as root for source and class files. It is useful to have this option set as the default for all projects. **Do so now** by following the `Configure default` link on the right and selecting `Project`. Click `Finish` to complete the project creation process.

A project folder will be added to your workspace, with the new project displayed in the `Navigator` view pane.

Adding Java code to the project.

A Java program is a collection of classes. Each class is defined in a separate `.java` source file and later compiled into a `.class` file. **A .java file must have the same name as the class it defines**.

We'll demonstrate how to add the class definition presented in Figure 1 to the project. First, make sure the project is selected in the `Navigator` view by clicking on it. Then select `File>New>Class` from the menu. Enter "Hello" into the `Name field`. Click the check box indicating that you would like Eclipse to include a **public static void main(String[] args)** method. Then click `Finish`.

A Java editor for `Hello.java` will open. In the main method enter code as shown on figure 1.

```
/**  A very simple Java program
      Created on 1/20/10 by W. Lucas
*/
public class  Hello {

  public static void main (String args[]) {
    System.out.println("Hello, world!");
  }
}
```

Figure 1: The *Hello* program.

Save by pressing Ctrl-s or by clicking on the Save button. This automatically compiles `Hello.java`.

The Eclipse editor includes very useful features such as syntax highlighting, parenthesis matching and much, much more. One of the most useful ones is the **auto-formatting feature**, which modifies the Java code in the editor window so that it is properly *indented*. As we will learn, consistent indentation is one of the hall-

marks of professional-looking code, as it makes the code much easier to understand. Pressing `Ctrl-Shift-F` or selecting `Source > Format` will format the selected code for you.

Compilation and Syntax errors

Eclipse compiles a Java file each time it is saved. If the compiler detects syntax errors in the program, error messages will appear in the `Problems` view pane on the bottom of the window. Each line containing an error will also be marked in the editor pane by a red circle with an X inside it. To experience syntax error detection, erase the semicolon at the end of the `System.out.println` statement and save the file. Place the cursor on the displayed error mark and read the error description that appears.

Sometimes a red error mark on the border of the editor window will also have a yellow bulb displayed next to it (a `Quick Fix` sign). This designates that Eclipse has suggestions about how the error could be fixed. To see the suggested fixes, click on the `Quick Fix` sign. To make Eclipse carry out its suggestion, simply select it by a mouse click. To see how the `Quick Fix` feature works, rename the class from `Hello` to `Mello` within the code. The `Quick Fix` list should contain at least two options: renaming the compilation unit (i.e. the file) or renaming the type (i.e. the class), so that the file name matches the name of the class. Pick the second option.

Running the program

Click the `Run` button (a green circle with an arrow) and the Hello program will run. Notice the `Console` view appearing in the bottom window. This view shows the output from running the program and also allows the user to enter input data as necessary. At this time, the `Console` should display the following string:

```
Hello, world!
```

Creating a project for an existing java file

Here's how to make your project refer to a java file that already exists, such as one that has been posted to the course website and then downloaded to your computer. In Eclipse, select `File>New>Java Project`. In the next window, specify the name of the project and click on the `Finish` button. Then, to add the existing file to your project, perform the following steps

1. select the project in the `Navigator` view
2. `right-click` and select `New>File`
3. click on `Advanced` and select the `Link to a file in the filesystem` option
4. click on the `Browse` button and then find and select the java file you are importing
5. add the name of the file to the `File Name` field if it isn't already there; don't forget to specify the extension .java

A simpler approach is to create a folder in your workspace and copy the java file into that folder. Then select `File>New>Java Project`. Enter the name of the folder you created as the Project name and press the `Finish` button.