CST316 Software Enterprise II: Construction and Transition **Lab: IDE Module**

Spring 2016

Objectives

1. Understand how to run Eclipse, configure the build system, and manage Eclipse workspaces
2. Create, build, and execute a simple Eclipse Java project
3. Use Eclipse with an external library
4. Apply, modify, and run Ant build scripts

TASK 1: Create and Execute a Simple Project in Eclipse

Begin this task by installing Eclipse. We have provided standalone versions of Eclipse and related tools for this course, a link is provided on the class website.

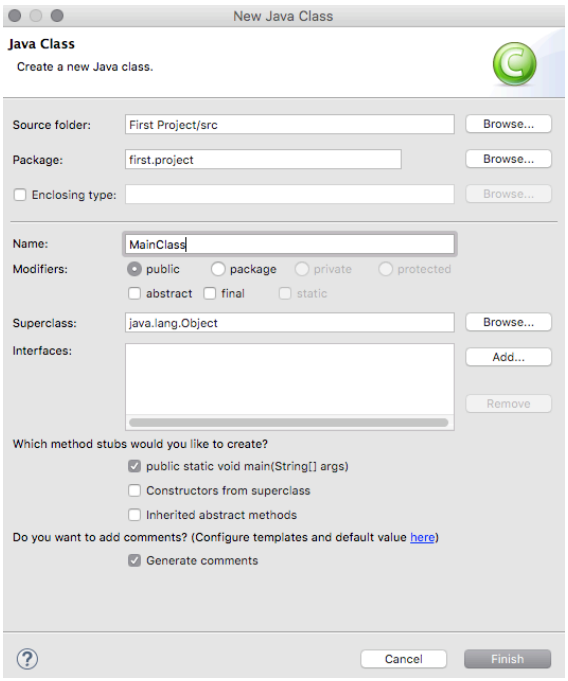
Start Eclipse; it will prompt for a workspace. Create a workspace in a directory where you would like to keep your working files. For this class I suggest a workspace folder that indicates it is for cst316. For example, I name my cst316 workspace "workspace316".

Configure the Eclipse Java compilation environment

Open Window (Eclipse) > Preferences and select Java > Build Path. Select the radio button "Folders" and set the source folder name to 'src' and the output folder name to "classes". These are the directories used by the Java builder to compile code located in 'src' and place the binary code in 'classes'. Next, tell Eclipse where your Java JDK is located by selecting Java > Installed JREs. If your JRE is not listed, right click on Add, select "Standard VM", click Next and browse to the home directory of your JDK (note: you should have Java 8 for this course). Finally select Java > Compiler in the Preference and set 'Compiler compliance level' to '1.8'. Select 'OK' to close the Preferences dialog.

Create a project

Create a project by right-clicking in the Package Explorer window and select New > Project. Eclipse starts a wizard that guides you through the construction for several types of projects. Select 'Java Project' and click Next. Name the project 'First Project' and create the project in the workspace (the default). Click Next which will create the project and open the project-specific settings. Notice Eclipse creates src and classes directories based on defaults set in the workbench preferences. They can be changed now or anytime later as project-specific preferences. Click Finish.



Compile, build, and execute Java code

Expand the First Project project in the Package Explorer. Create a Java source file by right-clicking on the 'src' package and selecting New > Class. Fill in the dialog box as shown below setting the package name to 'first.project', the class name to 'MainClass' and checking the creation of the main() method and Generate comments as show in the picture below.

Then click Finish. Clicking Finish should open the new class in the Java Editor. Note that you should see a comment with "TODO" as the first line of your main method, and this line has a *marker* displayed in the left margin. Cut-and-paste the code shown below and save the file (File > Save). Note the marker goes away.

```
package first.project;
public class MainClass {
   public static void main(String[] args) {
      String message = "Hello World";
      int x = 10;
      System.out.println(message + " " + x);
   }
}
```

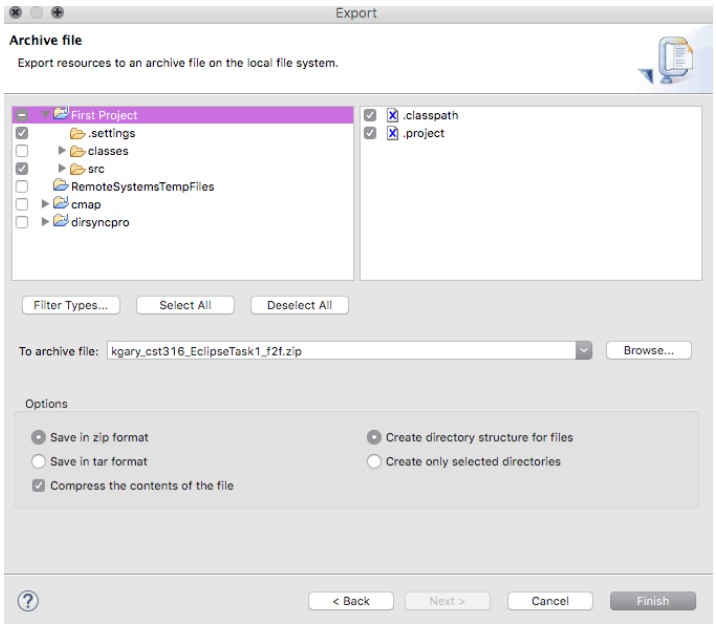
By default, Eclipse incrementally builds your source code using rules specified but the project's builder. To verify the code is already compiled, open the Navigator view by selecting Window > Show View > General > Navigator. Drill into the classes directory to observe the .class file. When finished click the 'x' to close the Navigator. Execute the program by right-clicking on the MainClass and selecting Run As > Java Application. Observe the program output in the Console window.

Next, execute the program in the debugger. First set a breakpoint on the main() method by right-clicking in the grey bar of the Java Editor next to the main() method and selecting Toggle Breakpoint. A blue dot should appear next to the line indicating the breakpoint.

Rerun the program in the debugger by selecting Debug As > Java Application. Eclipse will automatically prompt you to switch to the debugger perspective. Select Yes and notice the default views in this perspective. We continue to have our editors and Outline view, but have added Debug, Variables, and Breakpoints views. Single step through the execution by clicking the 'step over' button. Observe the added data in the Variables view after each execution and finally the program output in the Console view. Select the play button to run the program to completion or the stop button to terminate. You can restart the execution by right-clicking the MainClass Java Application in the Debug window and selecting 'Relaunch'. You can exit debugging and return to the Java perspective by selecting it in the upper-right.

Submission

To submit this task, use File > Export. Choose General > Archive. You will see an Export dialog, select the First Project (make sure any other projects you may have in Eclipse are deselected). Export the .settings and src directories by making the proper selections, see the figure at right. Note the name of the archive file, and the choice of "Save in .zip format".



I suggest testing your export by performing a corresponding Import (this is how we will grade this task). Go to File > Import, then General > Existing Project into Workspace, then Select archive file. Browse to your zipfile and import it. Note that there is a General > Archive File under Import, but this is not the correct option for importing a project from a zipfile (confusing I know).

TASK 2: Using Projects with Library Dependencies and Build Scripts

In the context of this course you will be working with multiple source files, configuration files, and external library dependencies. A benefit of an IDE is in the ability to help you manage the dependencies, run the scripts, and provide a nice view of all of your resources. In this task you will configure an existing Eclipse project we give you to use an external library and an existing Ant build script.

Step 1: Download the task2 zipfile from the class website. This is an Eclipse project archive. Import the project from the archive file. If done correctly, you should now see the project "GUIProject" in your Package Explorer. Browse the contents of the project; note its directory structure.

Step 2: You should notice when you loaded the project that there are red markers under the src directory in a couple of places? BarChartFrame has a dependency on a class BarChart, but there is no source code for BarChart.java. Instead, the barchart.jar file in the lib directory has this class. Use a command-line shell and the "jar tvf" command to discover the contents of this jarfile. Then, modify the Build Path of the project to include the jarfile, by right-clicking on project GUIExample > Build Path > Configure Build Path. Add the jarfile under the Libraries tab. If done correctly, the compile errors should go away.

Step 3: Run the application. Right-click GUIExample again and select Run As > Java Application.

- What happens? Why did the program not work?
- Inspect the main program, what is it expecting?
    - Now go to Run > Run Configurations and select your Main configuration. Using the arguments tab, add the missing reference to graph.txt
- Run the program again? Did it work? No, not yet?
    - Now you have a different problem; the graph.txt file needs to be in the runtime classpath for Java to be loaded as a resource. But the file is in the resources directory, which is not currently on the classpath. Go back to the Run Configuration, select the Classpath tab, highlight User Entries and select Advanced to find the option to add the resources folder.
- Now you should be cooking with gas!

Step 4: So what happened above was we had to configure Eclipse to compile with an external library dependency (Step 2), and tailor the runtime invocation (Step 3). This is akin to setting the classpath for the javac and java command-line programs in Java. But as we will discuss in class, anything beyond a simple program usually requires more than source compilation, it requires compilation, assembly, and arrangement of resources package for execution. To this end we use a build scripting tool, in our case Ant. In this GUIExample project, there is an Ant script with corresponding configuration files (build.xml and build.properties respectively). Open these files and inspect their contents and understand what they do. In particular notice in the run target how ant solves the graph.txt problem. Then:

1. On a command-line, execute the ant compile and the ant run tasks to compile and run the program. Then execute ant cleanall, and re-do these commands.
2. Go back into Eclipse. If you opened the build.xml before you should see the Ant view on the right of your screen with the ant icon and "SimpleGUIProject" (the value of our project name attribute at the top of build.xml). Expand the icon and you see all of the ant targets listed. Double-click run and see Ant run the program.
3. Edit the build.xml to add a target named "dist". Make the dist target create a jarfile of the entire project directory. See the Ant documentation for the jar task to see how to do this. The jarfile should be named according to the dist.jar property in build.properties.
4. Either at the command-line or in Eclipse, execute the "dist" target to create a jarfile for part of your submission (Note we will use this instead of Export for submission of this task).