Lab 2 – Using Eclipse

Contents

[1 Introduction 1](#_Toc166936584)

[2 Hello World 2](#_Toc166936585)

[3 Using the Debugger 7](#_Toc166936586)

[4 Auto-Generate Code 15](#_Toc166936587)

[5 Commenting Out Code 17](#_Toc166936588)

[6 Refactoring: Renaming Variables, Methods, Classes, Packages 18](#_Toc166936589)

[7 Copying Classes & Packages 19](#_Toc166936590)

[8 Resetting IntelliSense/Autocomplete 20](#_Toc166936591)

[9 Submitting your Work 21](#_Toc166936592)

[Appendix 1 n/a 22](#_Toc166936593)

# Introduction

The objective of this Lab is to learn the basic features of Eclipse to do Java development. You will learn: (a) how to create packages and classes and run them, use the debugger to set breakpoints and step through code, (c) how to autogenerate getters, setters, constructors, and *toString*, (d) quickly comment out a selection of code, (e) use refactoring to rename variables, methods, classes, packages, and projects, (f) copy classes and packages, (g) use IntelliSence and autocompete, and (h) how to zip up your work for submission. If you want additional resources, these are some tutorials on Eclipse:

* <https://courses.cs.washington.edu/courses/cse373/13wi/eclipse-tutorial/index.shtml>
* <https://www.tutorialspoint.com/eclipse/index.htm>
* <https://www.vogella.com/tutorials/Eclipse/article.html>

Finally, IntelliJ is another popular IDE for Java development. We will not cover IntelliJ; however, here are some tutorials:

* <https://courses.cs.washington.edu/courses/cse373/22su/projects/cse143review/intellij/>
* <https://www.tutorialspoint.com/intellij_idea/index.htm>

# Hello World

In this stage we will create a *HelloWorld* app and along the way learn how to write code using Eclipse.

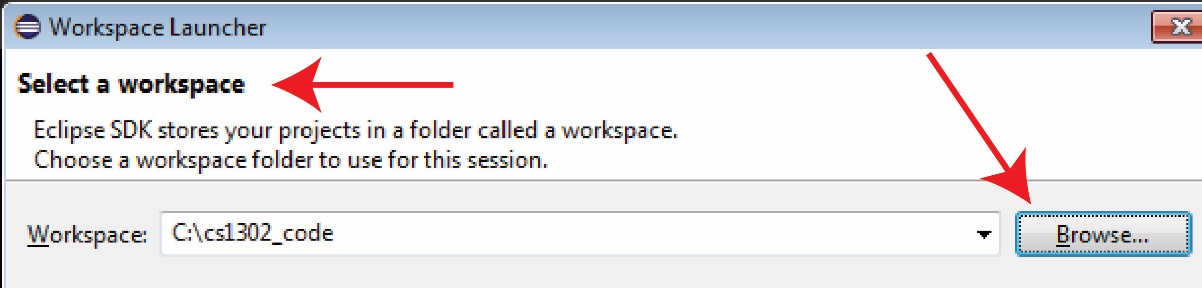
1. (Read, no action required) Basic information about using Eclipse:
2. Two important definitions:
3. *Workspace* – A folder on your computer where you store *projects*. Each time you launch Eclipse, it asks you for a *workspace* folder. You will create this folder using File Explorer.
4. *Project* – A folder inside a workspace where you store code for a particular assignment (*e.g.* lab or hw). A workspace you can have any number of *projects*. A project is created by Eclipse.
5. Next, we discuss several ways to organize your code for this class:
6. Use one workspace, *e.g cs1302\_code* for all the work you do in this class. Inside this workspace you would have *projects* with names like: *hw1, hw2, …, lab1, lab2, …*.
7. Create a separate workspace for each lab, homework, etc. For example, you might have a *lab01\_code* workspace that contains a *lab01* project, a *lab02\_code* workspace that contains a *lab02* project, *etc.*
8. Some variation of above. It is completely up to you. And, the name for the workspace(s) is completely up to you.

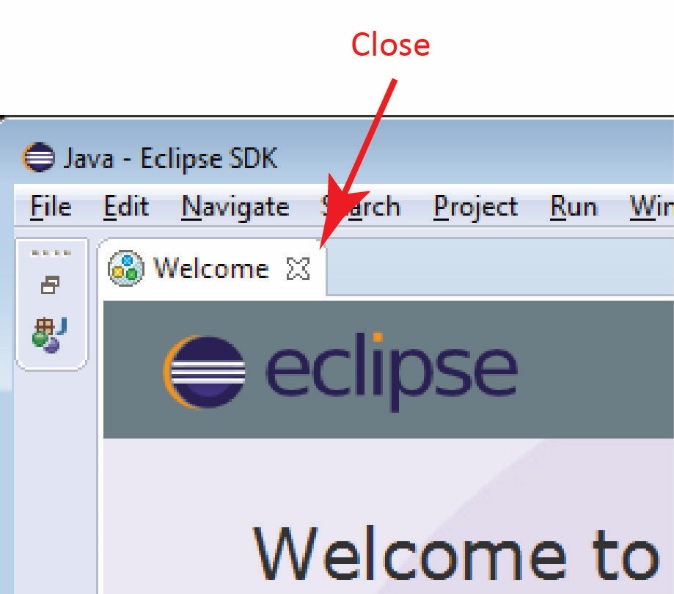
Note: I use the suffix “\_code” to indicate that such a folder is a workspace.

1. Next, I’ll give you a sense of how I use workspaces to manage all the code that I maintain for this course. I have the following workspaces for this course:
2. An individual workspace for each chapter (*e.g. ch09\_code, ch10\_code, etc.*)with projects for each example and practice problem for that chapter
3. One workspace for all the homework solutions (*e.g. hw\_code*), with projects for each homework: *hw01\_solution, hw02\_solution, etc.*
4. One workspace for all the test solutions (*e.g. test\_code*) with projects: *test1, test2, etc*.

Again, do whatever makes sense for you.

1. **Create your workspace** – Do the following:
2. Create a workspace, *i.e. c*reate a folder on your hard/flash drive. For example: *cs1302\_code* (or *lab02\_code,* or *labs\_code, etc*).
3. Launch Eclipse by double-clicking an icon on the desktop that is similar to the one shown on the right. (Yours will probably say something like: *eclipse-2023-06*)
4. As Eclipse opens, it will ask you to select a *workspace* as shown below. Choose: Browse and navigate to the folder you created above for your workspace (or paste the location into the textbox).

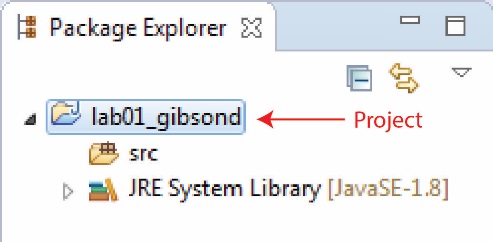


1. You will probably see a Welcome Screen. Close it if you want to.
2. **Create a project** – Do the following:
3. Choose: File, New, **Java Project**

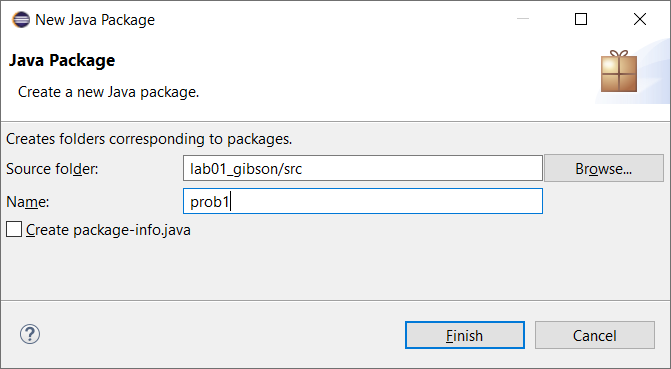
|  |
| --- |
| Note: In the off chance you do not see: *Java Project,* then try this:  Choose: File, New, Project, Java, Java Project, Next |

1. Supply the name, *lab02\_lastname, e.g.* *lab02\_gibson.*
2. Choose: Finish.

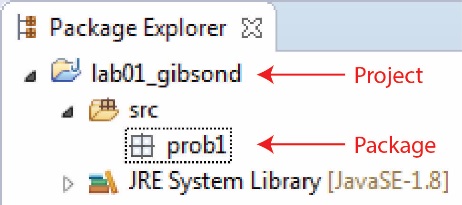
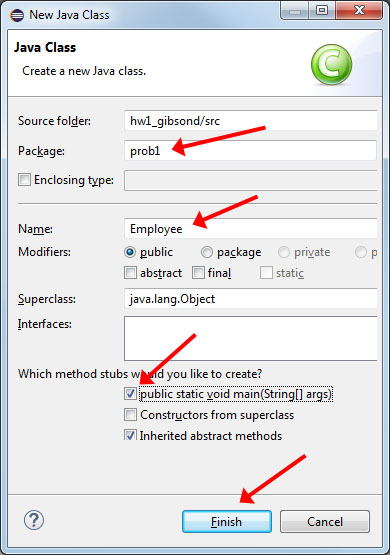
|  |
| --- |
| Eclipse Note: A *workspace* can contain any number of *projects*. Thus, as we said before, you can create one workspace and have a separate project for each lab, homework, *etc.* Or, as we said before, if you are creating a separate workspace for each assignment, then you will probably have just one project in each workspace. However, I frequently have several. For example, if I’m going to add something tricky to an existing piece of code, I would first copy the project and give it the name: *myproj\_ver1*, and now that is the backup, and *myproj* is the current one that I will modify*.* Before the next change to code, I’d copy it to *myproj\_ver2*, *etc.* A much better way to go about this is to use GitHub to store your projects. You will learn that in CS 4321 and other courses. |

1. Find the Package Explorer on the left side of the window and expand the project node (*lab01\_gibson*) and verify that the Package Explorer looks similar to the figure on the right (yours will say, *lab02…*). Next, we create a *package* inside *src* node and that is where the code you write will reside.

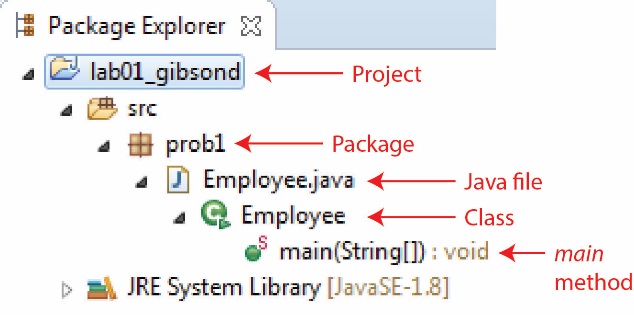
If you do not see the Package Explorer, then choose: Window, Show View, Package Explorer.

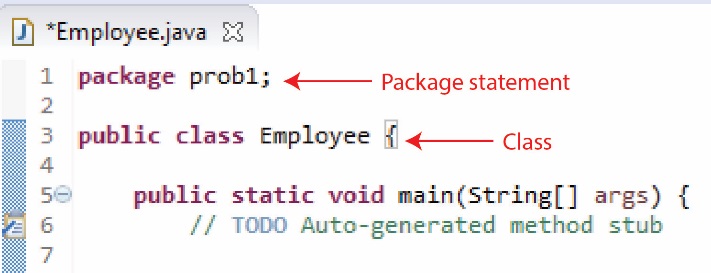
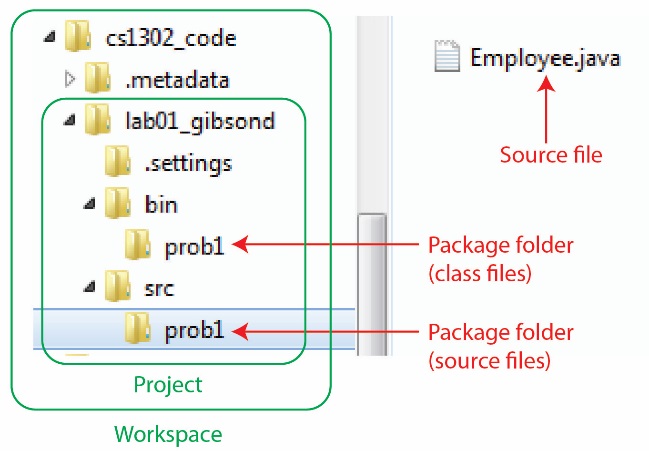
1. **Create a package** – Do the following (in the figure, yours will say, *lab02…*):
2. Select the project node in the Package Explorer (or the *src* node)
3. Choose: File, New, Package
4. Supply the name, *prob1* (lower-case)
5. Choose: Finish.

|  |
| --- |
| Java Note: A *package* in Java is way to keep code organized. Each homework and lab will have several problems for you to work. You will create a package for each problem. We will discuss packages more at some point in the course. |

1. Verify that the Package Explorer looks similar to the figure on the right (yours will say, *lab02…*).
2. **Create a class** – Do the following (for “Source folder”, yours will say, *lab02…*):**
3. Select the *prob1* package node in the Package Explorer.
4. Choose: File, New, Class (see figure on right).
5. Verify that the Packageis *prob1*
6. Supply the Name: *Employee*
7. Select the check box to generate *main*.
8. Choose: Finish.
9. Expand the nodes in the Package Explorer and verify that it looks similar to the figure below.

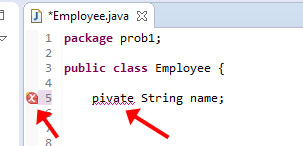
**🡺 Take a minute to study and understand how the Package Explorer organizes your code.**



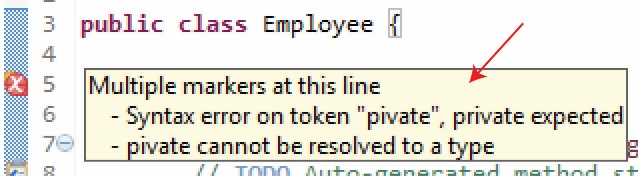
1. Verify that the Code Window looks similar to the figure on the right.
2. Leave (but don’t close) Eclipse and open File Explorer. Navigate to your workspace folder. Verify that it looks similar to the figure on the right.

**🡺 Take a minute to verify how your workspace and project are stored on your hard drive.**

We will not utilize these folders: *.metadata, .settings, bin*.

1. **Add some code** – Do the following:
2. Copy this (incorrect) instance variable below and place in the *Employee* class (do not put it in *main*)

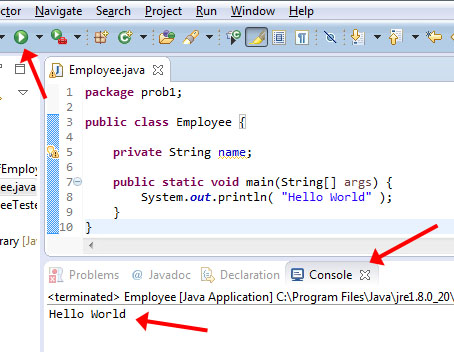
pivate String name;

1. **Read (no action required)** – The code is compiled as you type. You’ll see a little red X in the left margin when there are compile errors and red squiggly lines under the problem.
2. Hover your mouse above the red X to see what the compile error is.

**🡺** Note: You can right-click on the X and choose: Quick Fix to fix **some** errors. You can try on this one but none of the suggested solutions is correct.

1. Fix the error: *i.e.* change “pivate” to “private”.
2. Add this line to *main*:

System.out.println( "Hello World" );

1. **Run Code** – Do this:
2. There are four ways you can run your code, choose one:
3. Click the green arrow in the Toolbar as shown in the figure on the right
4. Choose: Run, Run
5. Press Ctrl+F11
6. Right-click in the source code and choose: Run As, Java Application.

* You may be presented a dialog asking how you want to run the file. If so, choose: “Run as Java Application.”
* You may be prompted to save your file

1. You should see “Hello World” in the Console window at the bottom as shown in the figure above.
2. Close: *Employee.java*.

# Using the Debugger

In this stage you will learn to use the debugging features of Eclipse. A debugger allows you to execute your code one line at a time. Thus, you can execute one line, then observe the value of variables, execute the next line, observe the value of variables, *etc.* Note:

* **Using the debugger is a very important and must-have skill.**
* **Pay close attention to these steps. If you get lost, or something does not work correctly, just back up and start again.**
* **If you really can’t get something to work, please, just get the spirit of the tutorial. You do not need to contact me, just move on.**
* **After you complete the steps below, feel free to experiment with the debugger before moving on.**
* **Learn to use the debugger! When your code doesn’t do what it is supposed to do, you have to find WHERE it goes astray. The answer is to use the debugger (otherwise, the old-fashioned way of putting print statements throughout your code---messy!)**

1. **Create a new package** – Do the following:
2. Select the project node in the Package Explorer (*e.g. lab01\_gibsond*)
3. Choose: File, New, Package
4. Supply the name, *prob2* (lower-case)
5. Choose: Finish.

|  |
| --- |
| Note: This will (probably) be the last time we provide explicit instructions to create a package. From now on it will be, “Create a package named *prob2*.” |

1. **Create a class** – Do the following:

🡺 The code below will be new to you. You are not expected to understand it exactly, though you will probably get be able to tell what is going on from completing these steps. We’ll spend a lot of time, 2-3 weeks, in class talking about writing classes. For now, we are just learning to use the debugger.

1. Select the *prob2* package node in the Package Explorer.
2. Choose: File, New, Class
3. Verify that the Packageis *prob2*
4. Supply the Name: *Engineer*
5. Choose: Finish.
6. Replace the code in the *Engineer* class with the code below; however, do not remove the package statement at the top of the existing file in the code window in Eclipse:

**public** **class** Engineer {

// Instance variables

**private** String name;

**private** **double** hours[];

// Constructor

**public** Engineer(String name, **double**[] hours) {

**this**.name = name;

**this**.hours = hours;

}

// Getter for name instance variable

**public** String getName() {

**return** name;

}

// Method

**public** **double** getWages(**double** payRate) {

**double** totalHours = 0.0;

**for**(**double** h : hours) {

totalHours += h;

}

**double** wages = totalHours\*payRate;

**return** wages;

}

// Method

@Override

**public** String toString() {

**return** name;

}

}

1. **Read (no action required)** – Reference the comments in the class above and note the following.

* ***Instance variables***are variables we can use inside the class. They represent the ***state***of the class, the information that is important to keep around. In the example above, and engineer has a name and the hours that they worked.
* A ***constructor*** is similar to a method, except that it is used to create an *Employee* **object.** We will see how to create objects from classes shortly. The constructor accepts a name and an array of doubles, which represents the number of hours worked on each day and stores them in the **instance variables.**
* We use the ***this***keyword to differentiate the instance variables from the parameters that have the same name.
* The *getWages* method accepts a pay rate. It adds all the hours in the array and then multiplies it by the pay rate to get the total wages earned, and then returns it.

**Again, we will study these things in detail in class.**

1. **Create a class** – Do the following:
2. Create a class named *EngineerTest* in the *prob2* package and add the code below (see the previous two steps if necessary):
3. Replace the code in the *EngineerTest* class with the code below (do not remove the package statement at the top):

**public** **class** EngineerTest {

**public** **static** **void** main(String[] args) {

**double**[] hours = {8.0,8.0,8.0,8.0,8.0,0,0};

// Call the constructor to create an Engineer object named: “e”

Engineer e = **new** Engineer("Wilma", hours );

// Call the getWages method on e

**double** wages = e.getWages(20.0);

String msg = e.getName() + " made $" + wages;

System.***out***.println(msg);

}

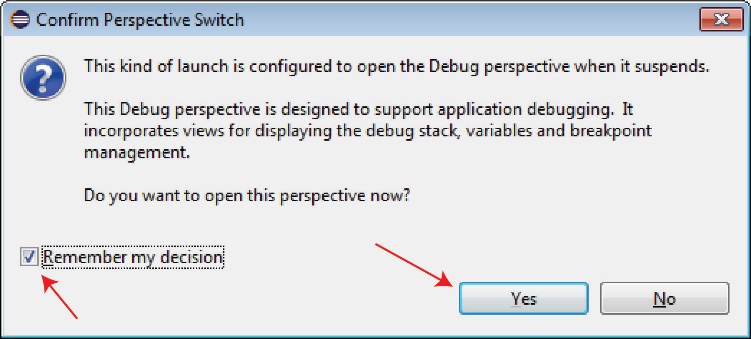
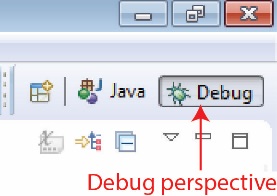
}

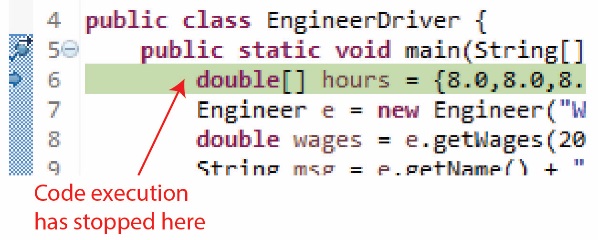
1. Run the program and observe the output
2. Power Tip – Select a method call, right-click, and choose:

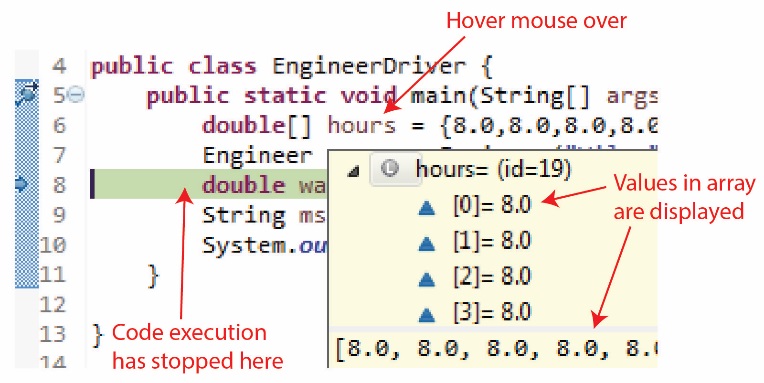
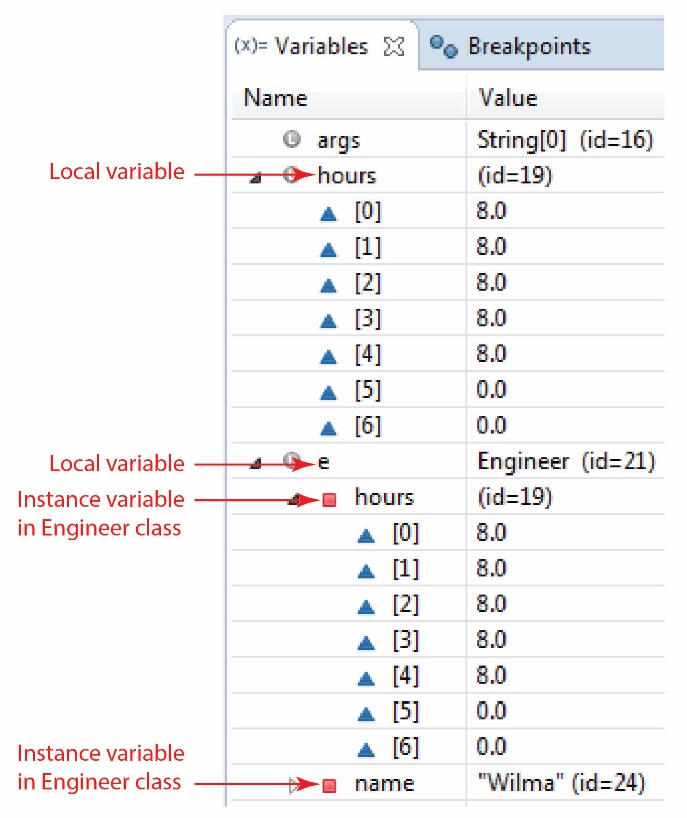
* **Open Declaration** – To display the method’s code. Do that now with the *getWages* method.
* **Open Call Hierarchy** – To show all the places a method is used. Opens a Call Hierarchy tab at the bottom of the screen. You can click on each entry to take you to a method call. Return to *Engineer Test* and do this with the *getWages* method.

1. Power Tip – A very powerful search is found by pressing: Ctrl+h (or choosing: Search, Search). Most useful are the File Search and Java Search tabs.
2. **Run the debugger** – Do the following:

|  |
| --- |
| Eclipse Note: As you go through the following steps you may lose track of where you are, get confused, *etc*. If so, simply stop debuggingC:\Users\dgibson\OneDrive - Valdosta State University\Data-Grant\Innovation2015\labs\01\e1.jpg by choosing: Run, Terminate (or press the red square on the Toolbar) and then start the section over. |

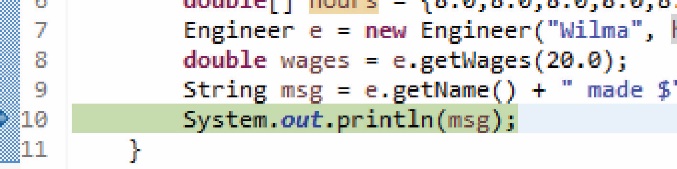
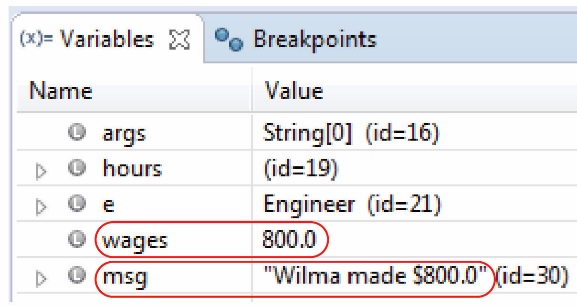
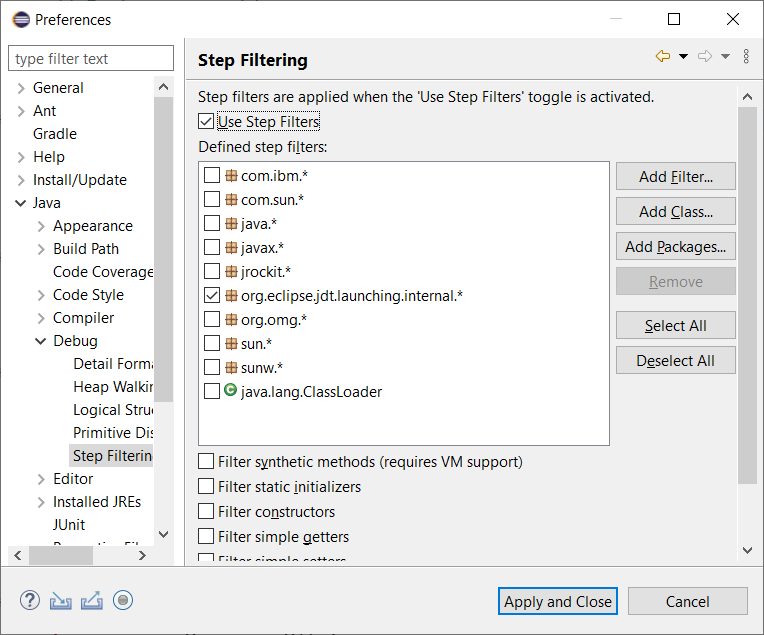
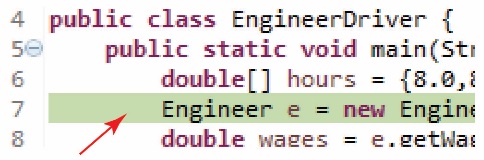
1. **Read (no action required)** – A *breakpoint* is a place in the code where execution will stop when running in Debug mode. You can set a breakpoint by double-clicking in the margin, to the left of the line number as shown in the figure on the right. A blue circle will be shown for each breakpoint.
2. Add a breakpoint at the line where the array is created in *EngineerTest* (the figure shows *EngineerDriver* instead of *EngineerTest* as yours will).
3. Run the debugger. Choose Run, Debug (or press F11) and answer “Yes” to the resulting dialog.
4. **Read (no action required)** – Note the following:

* The windows displayed have changed because now you are in the *Debug perspective* as indicated in the upper right corner of your screen (see figure on right. Yours will probably just show the symbol, and not the word, “Debug”). When we end debugging (later), we will want to go back and choose the *Java perspective* to the left of *Debug*.
* The green highlight (as shown on the right) indicates that the code execution has stopped here. The highlighted line has not been executed. When execution resumes, it will be the next line to execute.

1. Choose: Run, Step Over twice (or press F6 twice).
2. Hover your mouse over the *hours* variable (as shown on the right) and you will see the values in the array displayed.
3. Find the Variables window on the right of your screen and expand the nodes shown in the figure on the right. The Variables window shows all variables that are in *scope* and their corresponding values. See figure on right.

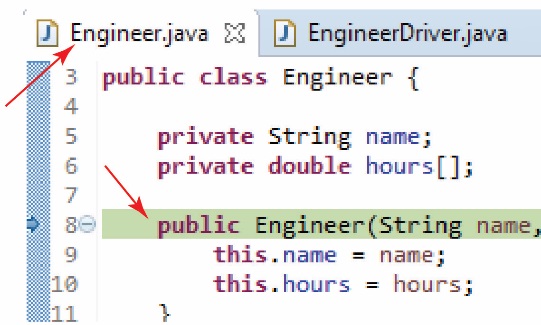
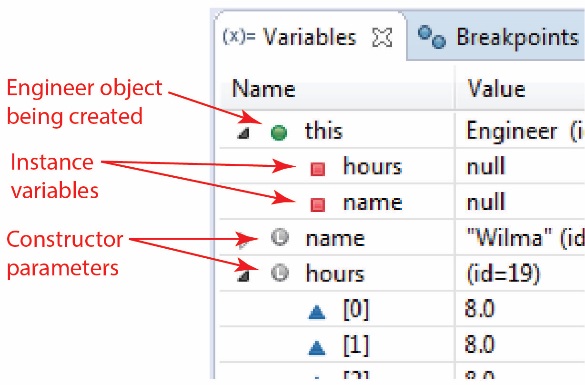
Note the following:

* *hours* is a local variable declared in *main.*
* *e* is a local variable declared in *main*. It refers to the *Engineer* object that we created on the preceding line.
* *e* has instance variables, *hours* and *name*. The constructor in the preceding line took the arguments and assigned them to the instance variables. Look back at the *Engineer* class’s constructor above.

1. Choose: Run, Step Over twice (or press F6 twice). Execution should be stopped at the last line of code.
2. Inspect the values for *wages* and *msg* in Variables window (you may need to collapse the nodes we expanded in the previous step). See figure on right.
3. Choose: Run, Resume (or F8) to finish the code execution.
4. The debugger has changed since I first wrote these tutorials. Below, I will have you step into the constructor. Unfortunately, that cannot be done on the first invocation of the constructor because the class is not in memory yet. A simple fix, which should be saved in your workspace is to do the following:
5. Choose: Window, Preferences, Java, Debug, Step Filtering
6. Check (at the top): *Use Step Filters*
7. Uncheck (at the bottom of the list): *java.lang.ClassLoader*
8. Choose: Apply and Close
9. Next, we will run the debugger again to illustrate some other features of the debugger. Do the following
10. Run the debugger: choose Run, Debug (or press F11).
11. Choose: Run, Step Over (or press F6). You should be on the Engineer constructor (which hasn’t executed yet) as shown on the right.

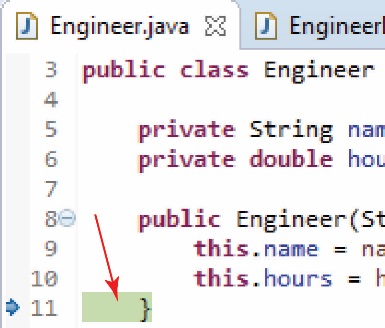
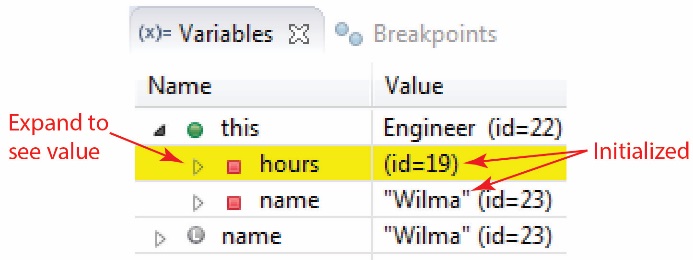
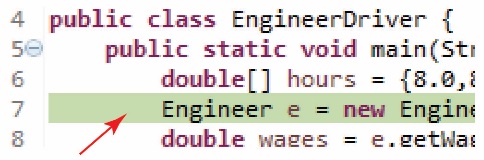
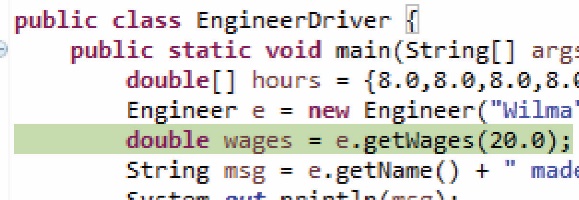
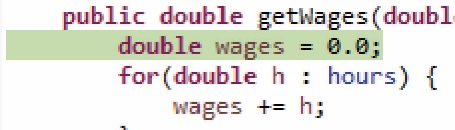
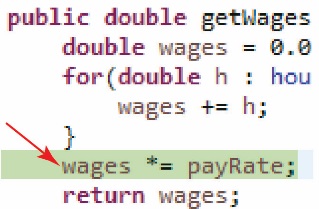
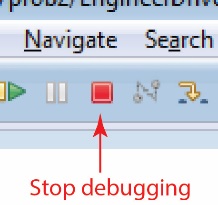
Note:

* The last time we ran the debugger we stepped over the line we are currently one (the constructor). Below, we will step into the constructor and then step through the code there. When you step over a method call (or constructor), the method executes completely and we proceed to the next line.
* When you step into a method you can step through the code in that method and when the method is complete, the debugger returns to the line of code that called the method.

1. Choose: Run, Step Into (F5). Notice that we have stepped *into* the *Employee* constructor as shown on the right.
2. Inspect the Variables window and note the items indicated in the figure on the right.

Note:

* *this* represents is *Engineer* object that we are creating.
* The instance variables have the value *null* because the code hasn’t executed yet to assign them the values of the parameters.
* The parameter values are shown.

1. Press F5 (or F6) three times. Execution should be stopped on closing brace in the constructor.
2. Inspect the Variables window and note that the instance variables are now initialized.
3. Press F5 (or F6) and execution should be returned to *main*.
4. **Press F5 (or F6) and execution should be stopped on the call to *getWages.*
5. Choose: Run, Step Into (F5) to enter the method. Execution should be stopped on the first line in the *getWages* method.
6. Press F5 six times, you will be going through the loop. As you do this watch the *totalHours* (variable is name *wages* in the figure below) variable in the Variables window as its value gets larger each time a value is added.
7. Suppose you want to run the rest of the code in the loop without having to step through it. Do the following:
8. Put cursor on line just outside loop. The line will be highlighted light-blue as shown on the right.
9. Choose: Run, Run to line (Ctrl+R). Execution should have advanced to that line.
10. Press F5 (or F6) two (or may need three) times and execution should be returned to main.
11. Stop debugging by choosing: Run, Terminate (or press the red square on the Toolbar).
12. Debugging will have ended but you will still be in the Debug perspective. Return to the Java perspective by pressing the Java button in the upper right of your screen. (Your display will probably not show the “Java” text on the button, only the symbol. If you hover over it, it will say, “Java”).
13. Close the open files: *Engineer* and *EngineerTest.*
14. Debugging Summary:

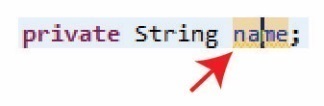
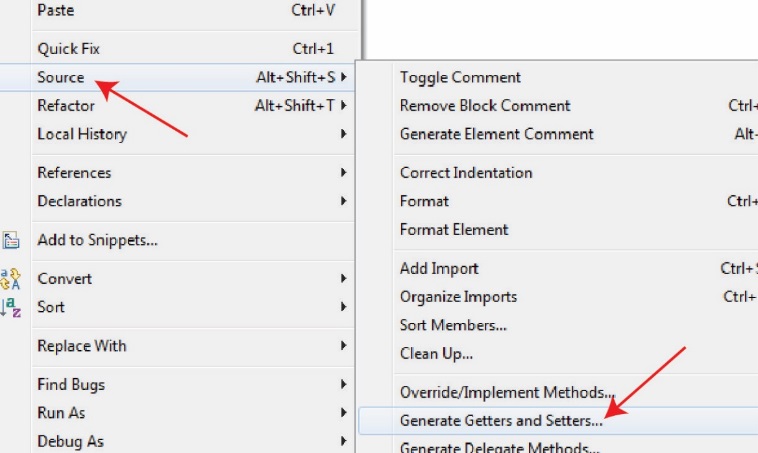
|  |  |  |
| --- | --- | --- |
| **Menu** | **Description** | **Short Cut(s)** |
| Run, Debug | Start debugging | Green arrow or F11 |
| Run, Terminate | Stop debugging | Red Square or Ctrl+F2 |
| Run, Step Into | Step into a method | F5 |
| Run, Step Over | Step over a method | F6 |
| Run, Run to Line | Run to a selected line | Ctrl+R |
| Run, Resume | Continue running to next breakpoint | F8 |
| Run, Remove all Breakpoints | Clear all the breakpoints |  |

# Auto-Generate Code

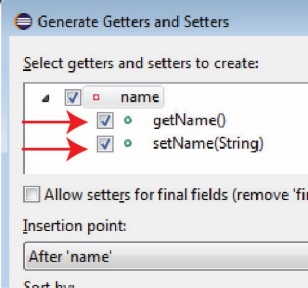
In this stage we will see how Eclipse can automatically generate certain code for us. For example, constructors, getters and setters, *toString*.

1. Do the following:
2. Create a package named *prob3.*
3. Add a class named *Employee* (*main* will not be needed here, so you don’t have to check that)
4. Add the following instance variable:

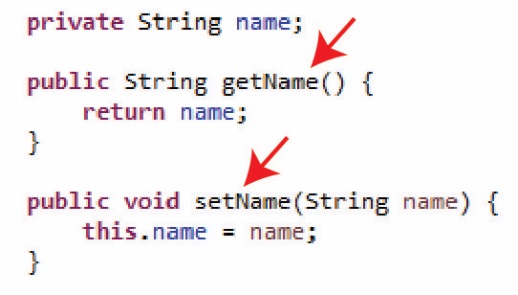
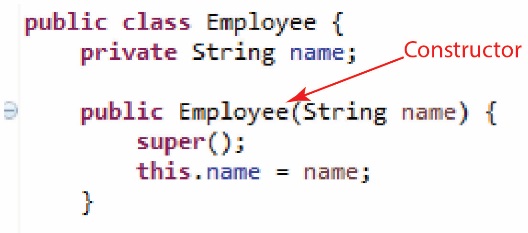
**private** String name;

1. **Generate getters and setters** – Do the following:
2. Click the *name* variable and it should highlight.
3. Right-click and choose: Source, Generate Getters and Setters…

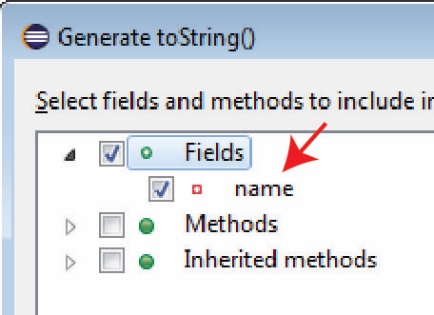
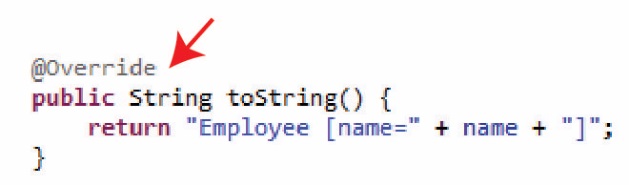
🡺 We will learn what getters and setters are in class.

1. Read the resulting dialog carefully to see the different options. Especially the *Insertion Point*. It can put the generated code in a poor location by default, so it can be useful to select where you want to add it.

At this point don’t change anything and then choose: Generate.

1. Verify that the *getName* and *setName* methods are present.
2. **Generate a constructor** – Do the following:
3. Right-click anywhere in the code window and choose: Source, Generate Constructor using Fields…
4. Choose: Generate
5. The constructor should be shown in the code window (shown in the figure on the right). We will learn what the *super()* line of code is in another chapter. It can be deleted, but fine to leave it.
6. **Generate *toString*** – Do the following:
7. Right-click anywhere in the code window and choose: Source, Generate toString()…

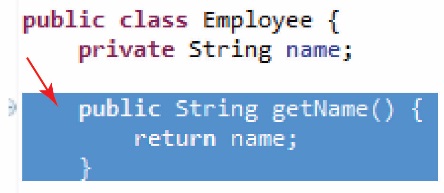
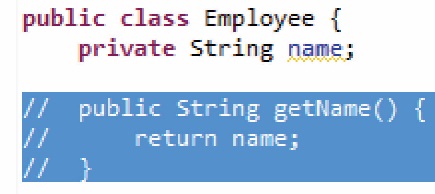
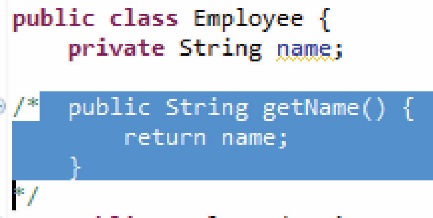
🡺 We will learn what *toString* is in class.

1. Read the dialog (partially shown on the right) carefully to see the different options. Note that you can choose which fields appear in the *toString*, among other options.Do the following:
2. Select the *Insertion Point*: “After ‘setName(String)’
3. Choose: Generate
4. The *toString* method should be inserted into the code as shown on the right. Eclipse adds the *@Override* annotation. We will learn what this means in another chapter. It is not required, but a best-practice, so leave it.

# Commenting Out Code

1. **Comment out a Block of Code** – Do the following:

**Note**

1. **Read (no action required)** – **This is a very useful technique for quickly commenting out a block of code and can be very useful when debugging.** Make sure you can do this!
2. Select the *getName* method (I’ve moved it to the top of the class so that the screen shots will be smaller).
3. Choose: Source, Toggle Comment (or press Ctrl+/). Line comments have been added as shown on the right.
4. Choose: Source, Toggle Comment again (or press Ctrl+/) and the line comments have been removed.
5. Make sure the method is still selected and choose: Source, Add Block Comment (or press Ctrl+Shift+/) and a block comment is created.
6. Choose: Source, Remove Block Comment (or press Ctrl+Shift+\) and the block comment is removed.

# Refactoring: Renaming Variables, Methods, Classes, Packages

1. Create a class named: *EmployeeTest* in the *prob3* folder and copy this code in (leaving the package statement at the top)

**public** **class** EmployeeTest {

**public** **static** **void** main(String[] args) {

String name = "Xavier";

Employee e = **new** Employee(name);

System.***out***.println(e);

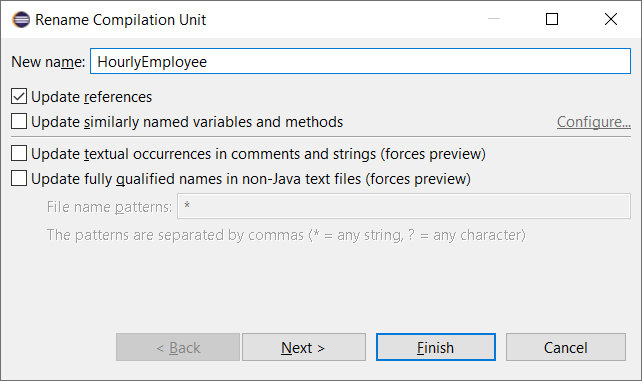
}

}

Verify that the program runs.

1. Next, we are going to change the name of the *name* instance variable in the *Employee* class. Do the following:
2. Open *Employee.java* if necessary and select the *name* instance variable (the declaration at the top, not in the code where it is used)
3. Choose: Refactor, Rename (or Alt+Shift+R)
4. A message will appear, “Press Enter to refactor”. Type: “firstName” and Enter and you will see that it changes all occurrences of *name* to *firstName*.

🡺 Note: This doesn’t (of course) change the name of the *setName* parameter, nor the name of the method itself.

1. We can change the name of a method, class, or package using the exact same technique.
2. Next, we are going to change the name of the *Employee* class. You can do this in the code window or you can do it in the Package Explorer. I usually do it in the Package Explorer. Do the following:
3. Save and close the files that are open (*Employee* and maybe *EmployeeTest*).
4. Select *Employee.java* (single-click) in the Package Explorer.
5. Choose: Refactor, Rename (or Alt+Shift+R).
6. A dialog appears, change the name to: *HourlyEmployee*
7. Choose: Finish

Note:

* It changed the name of the file to *HourlyEmployee.java*.
* Inside the class, it changed the name of the class and the name of the constructor. Verify this now.
* Inside the *EmployeeTest* class, it changed *Employee* to *HourlyEmployee*. Verify this now.

1. (Read, no action required) Always use: *Refactor, Rename (or Alt+Shift+R)*, to change the name of an instance variable, method, class, or package to ensure that your change correctly modifies all occurrences in any file in the package.

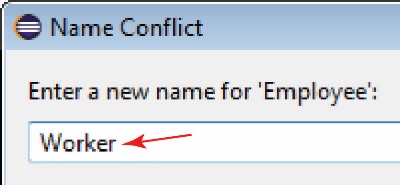
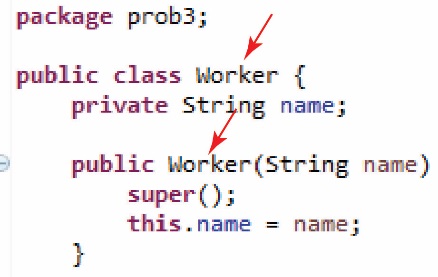
# Copying Classes & Packages

In this stage we show how to copy classes and packages in Eclipse.

1. **Read (no action required)** – In homework assignments and labs, I may sometimes ask you to copy a class or package. We do this so that your work is incremental, *i.e.* make something, copy and rename it, and then make some additions to the copy. Or, when I am working, I will frequently copy my package to simply make a backup. There are three scenarios we consider here:

* Copy a class and rename it in the same package.
* Copy a class from one package to another (and possibly renaming it).
* Copy a package and rename it in the same project.

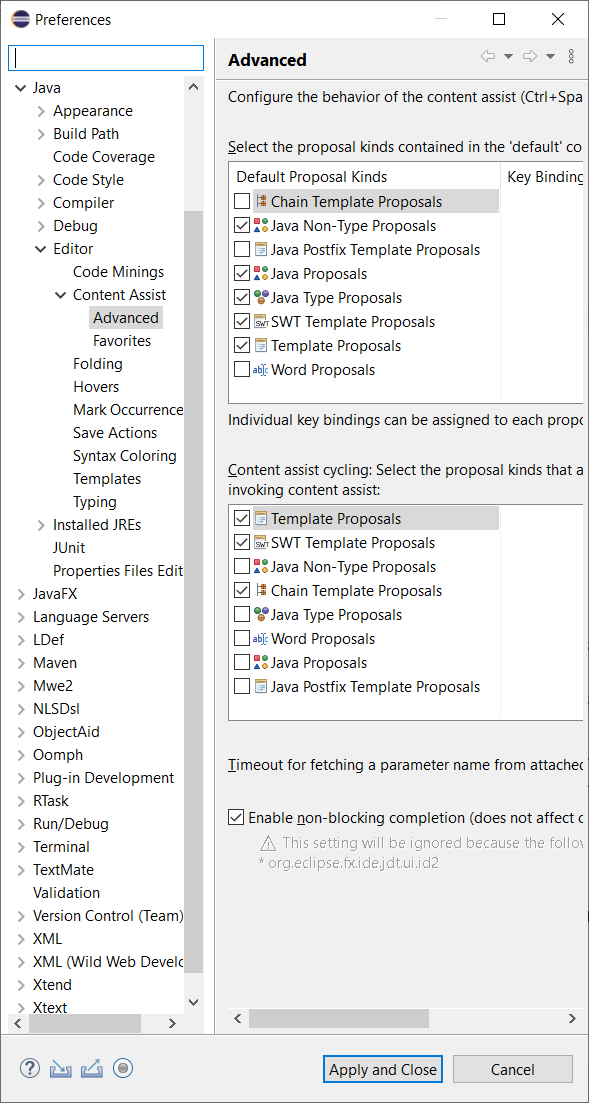
This is simple, but we need to point out a few things.

1. **Copy a Class in the same Package** – Do the following:
2. Save and close all files.
3. Select *HourlyEmployee.java* in the *prob3* package in the Package Explorer.
4. Ctrl+c (or choose: Edit, Copy)
5. Select the *prob3* package in the Package Explorer.
6. Ctrl+v (or choose: Edit, Paste).
7. In the resulting dialog, change the name to *Worker* and choose “OK.”
8. Open the *Worker* class and you should see that the name of the class has changed and the constructor name has changed also.
9. **Copy a Class in a different Package** – Do the following:
10. Select *Engineer.java* in the *prob2* package in the Package Explorer.
11. Ctrl+c (or choose: Edit, Copy)
12. Select the *prob3* package in the Package Explorer.
13. Ctrl+v (or choose: Edit, Paste).
14. Open *Engineer.java* in the *prob3* package in the Package Explorer and verify that the package statement has been changed to *prob3*.
15. **Copy a Package** – Do the following:
16. Select *prob3* package in the Package Explorer.
17. Ctrl+c (or choose: Edit, Copy)
18. Select the *src* folder in the Package Explorer.
19. Ctrl+v (or choose: Edit, Paste).
20. In the dialog that results, supply the name *prob4* and choose “OK.”
21. Expand the *prob4* package in the Package Explorer and open the files. Verify that these are the same classes as in *prob3* except that the package statement at the top of each file shows *prob4*.
22. **Read (no action required)** – You can drag a Java file from your computer directly into Eclipse however, you’ll have to change/add the package statement at the top. In homeworks (and labs) I will sometimes supply you with a Java file (zipped). You will download, unzip, drag the file into Eclipse, add (or change) the package statement.

# Resetting IntelliSense/Autocomplete

**Read (no action required)** – Sometimes, IntelliSense/autocomplete will stop working. Here is how you can fix it:

1. In Eclipse, choose: Windows, Preferences, Java, Editor, Content Assist, Advanced
2. Check these as show below: Java Non-Type Proposals, Java Proposals, Java Type Proposals

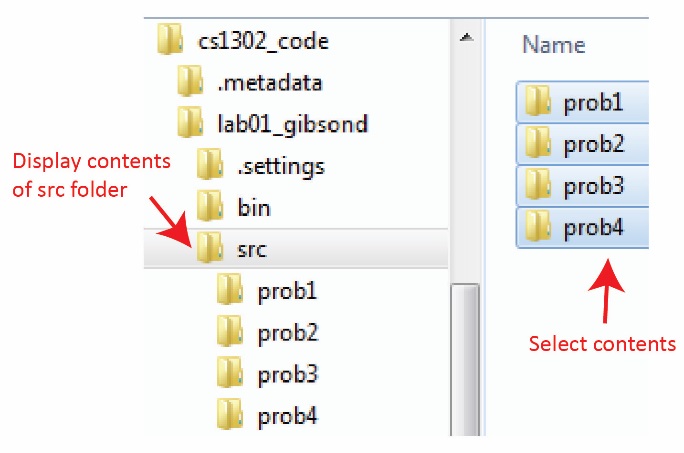


1. Choose: Apply and Close

Note: when I do this, it works, but frequently, the suggestion list will have every suggestion listed twice. Not sure why! Perhaps experimenting with the checkboxes above would correct that.

# Submitting your Work

All labs and homework will be submitted as described below, on Blazeview.

1. **Prepare for archiving** – Do the following
2. Make sure all your files are saved in Eclipse.
3. Although not necessary, I recommend closing Eclipse. Do that now.
4. **Archiving your work for submission** – Do the following:
5. In File Explorer, display the contents of the *src* folder for the assignment you are submitting (*e.g. lab01\_...*)
6. Select the contents of the *src* folder as shown in the figure on the right.

**🡺 Double-Check: Did you ONLY select the 4 *prob* folders? Not the *src* nor the *lab02\_lastname* folder.**

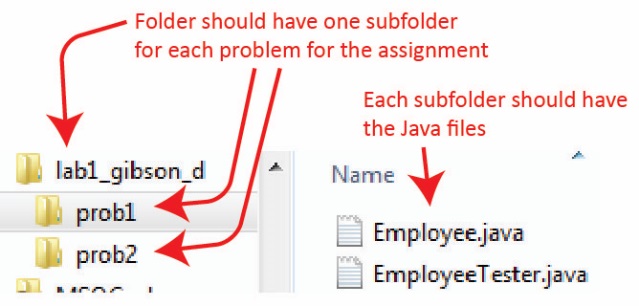
1. Right-click the selected folders and choose: Send to, Compressed (zipped) folder.
2. Provide the name:

*assignmentTypeNumber\_lastName*

For example: *lab1\_gibson* or *hw1\_gibson*

This will create a zip file with name: *lab1\_gibson.zip*.

1. (Read, no action required) I will take points off an assignment if you have not done exactly what I specified above. **In other words, your zip file must contain only: the CONTENTS of the *src* folder.**
2. Test your zip file – Do the following:
3. Copy your zip file from above to some new location (anywhere, but not in your workspace).
4. Right-click the file and choose: Extract All… and then choose: OK.
5. Verify that the proper folders and files are present.



1. Upload your zip file to Blazeview to the proper dropbox (in this case, Lab 02)

**You are done!**

Appendix

There is nothing currently in the Appendix.

1. n/a

.