

To use Eclipse you must have an installed version of the Java Runtime Environment (JRE).

The latest version is available from java.com/en/download/manual.jsp

Since Eclipse includes its own Java compiler, it is not strictly necessary to have a version of the Java Development Kit (JDK) installed on your computer.

However, I recommend installing one anyway so that you can test your code against the "real" Java compiler.

The latest version is available from: java.sun.com/

If you install the JDK, I recommend putting it in a root-level directory; my copy is installed in `E:\jdk1.6.0_14`.

Go to www.eclipse.org and click on Download Eclipse:

Get Started now...

Download Eclipse



Select the Eclipse IDE for C/D++ development:



Eclipse IDE for Java Developers (92 MB)

The essential tools for any Java developer, including a Java IDE, a CVS client, XML Editor and Mylyn. [More...](#)

Downloads: 920,162

Download the distribution:

Eclipse downloads - mirror selection

All downloads are provided under the terms and conditions of the [Eclipse Foundation Software User Agreement](#) unless otherwise specified.

Download eclipse-java-galileo-SR1-win32.zip from:



[\[United States\] OSU Open Source Lab \(http](#)

[BitTorrent](#) is available for this file.

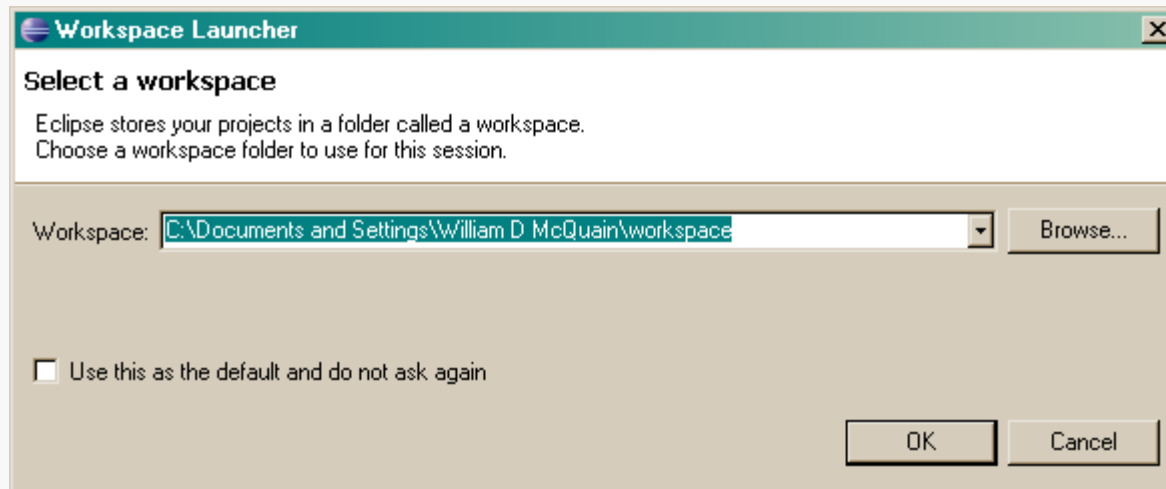
...or pick a mirror site below.

Unzip the distribution in an appropriate location.

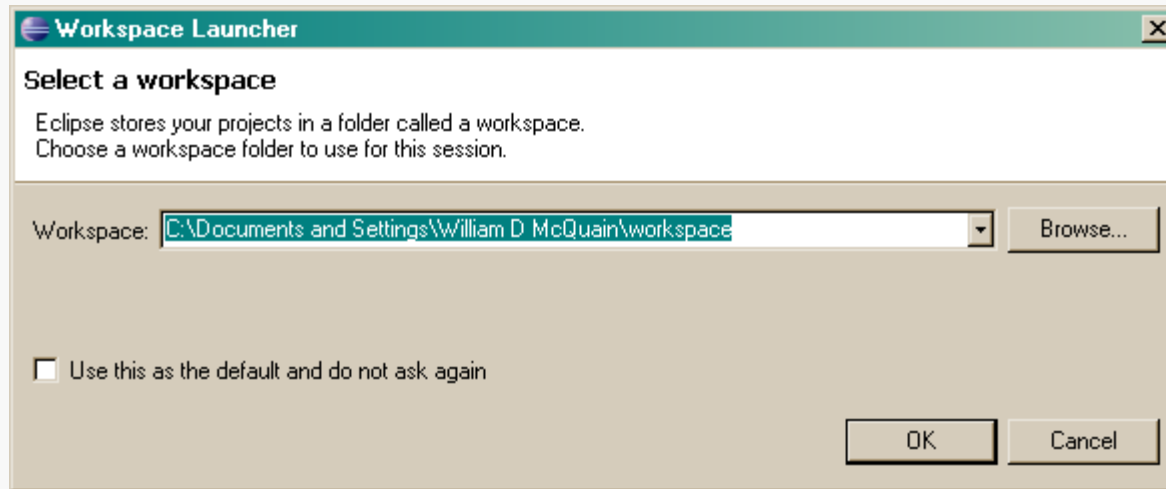
If you've already got another version, say Eclipse for C/C++ installed, I recommend placing this in a different directory tree.

In this case, I'm installing Eclipse to `F:\eclipse`. Note: the contents of the zip archive are already packed in a directory named `eclipse`, so I specified that the extraction should be to `F:\`. Avoid spaces in the path to Eclipse.

I find it useful to put a shortcut on my desktop; find `eclipse.exe` in the root directory of the installation and drag to the desktop to create a shortcut.

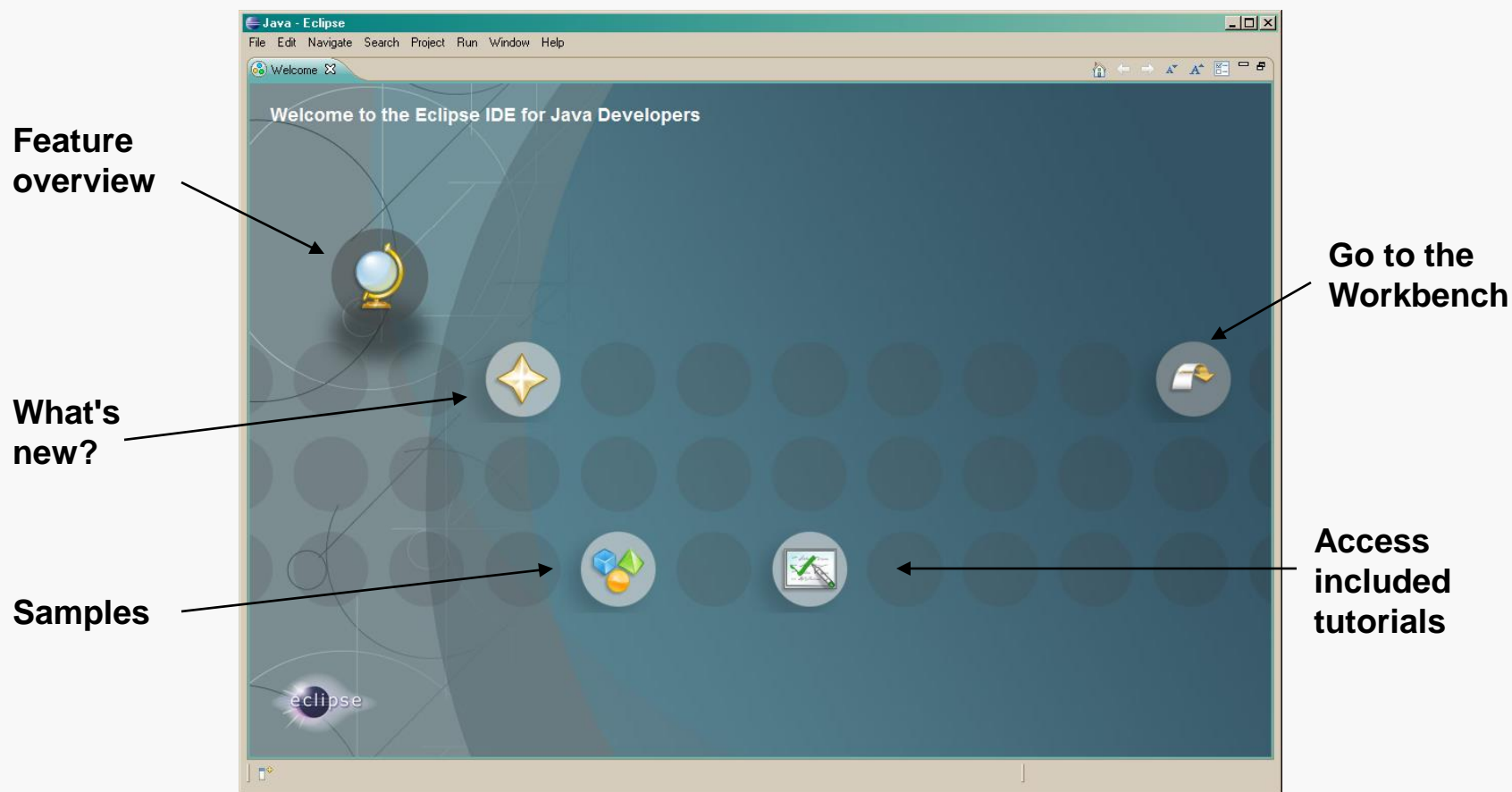


The first time you start Eclipse, you'll be asked to specify a location for the Eclipse Workspace; this is where Eclipse will, by default, keep your programming projects:



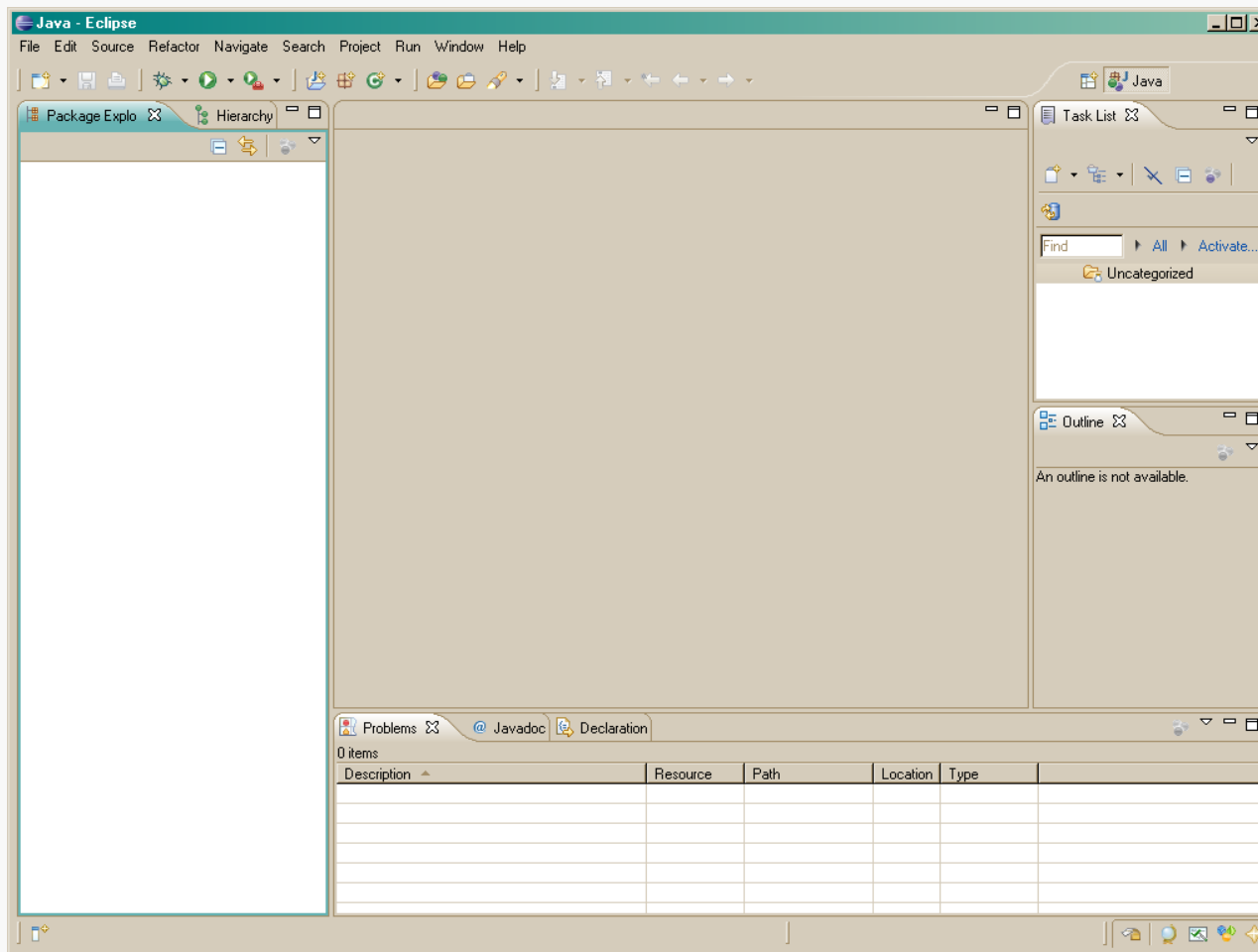
This can be anything you like. I set this to be `F:\JavaWorkspace` for my examples.

The initial startup looks like:



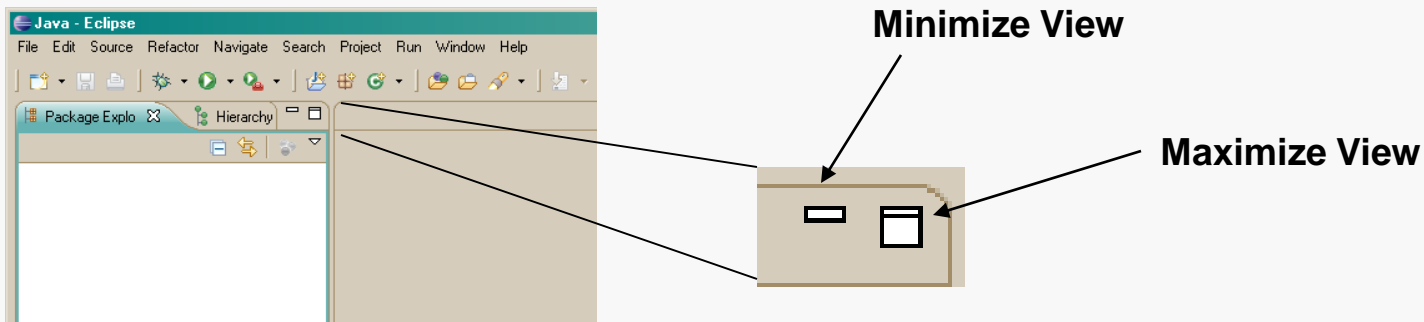
Feel free to explore the options offered here... I'm going to the Workbench...

The initial Eclipse workbench:



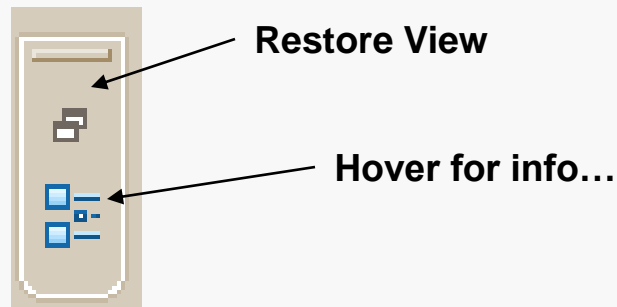
Eventually you'll want to use all of this, but let's clean it up a bit for a start...

The initial Eclipse workbench:

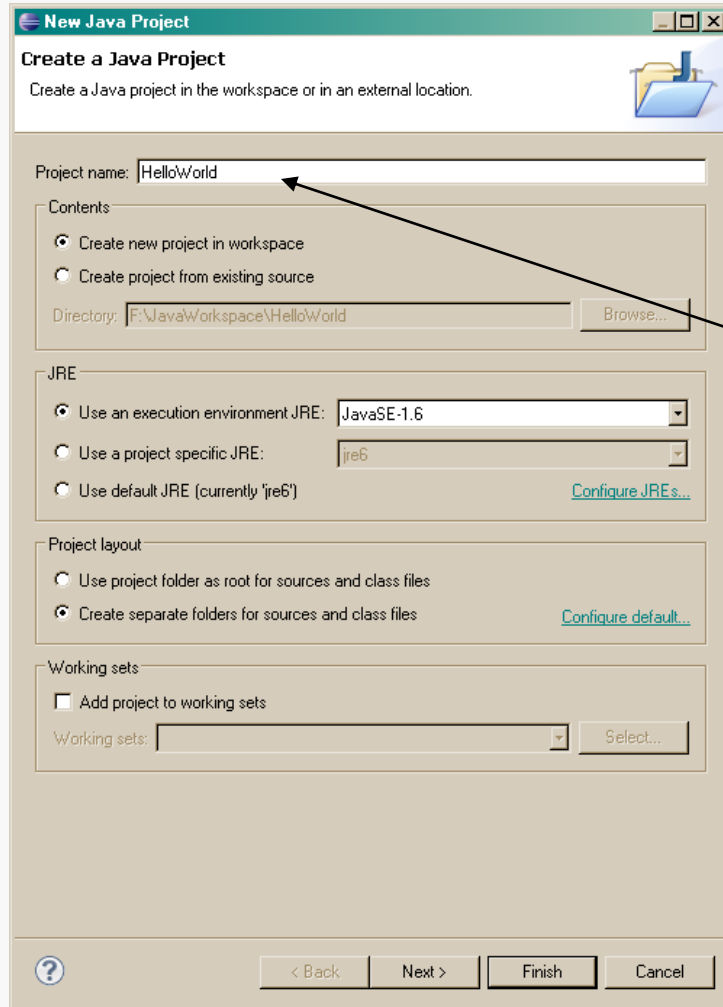
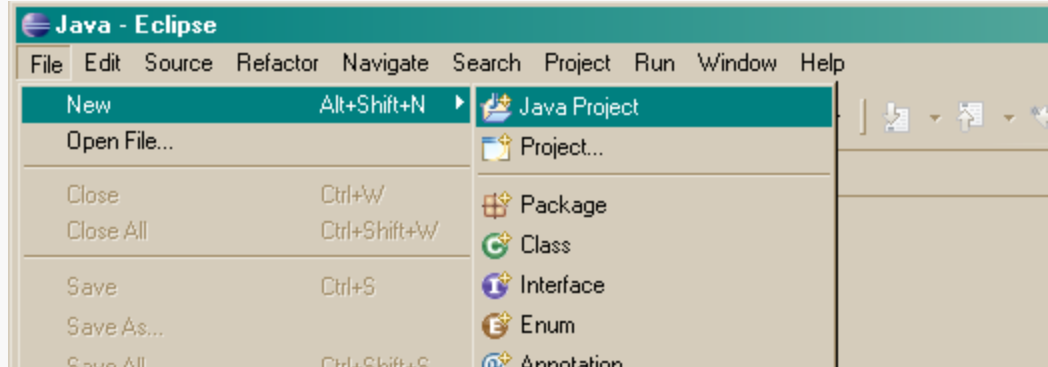


For now, I'll minimize the Task List and Outline Views on the right side of the Eclipse workbench, and the Problems/Javadoc/Declaration Views at the bottom...

Minimizing an Eclipse View reduces it to a graphic like that shown below:



In the Workbench, select File/New/Java Project:

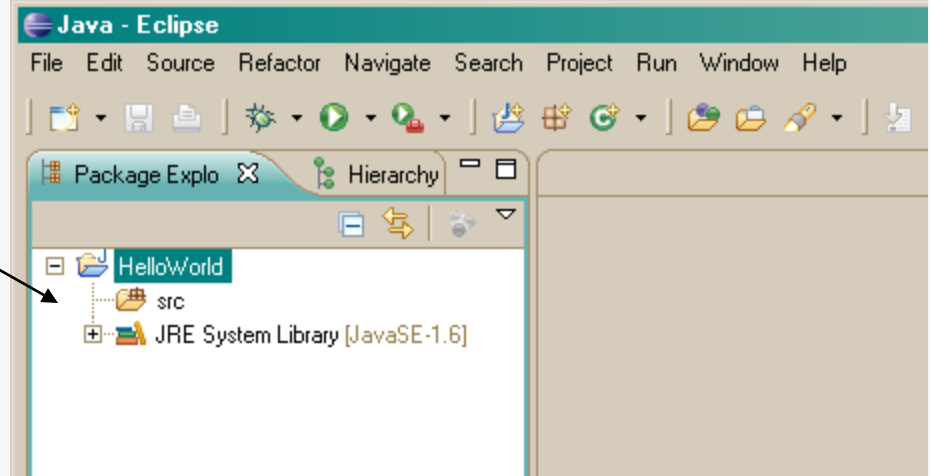


Enter a name for the Project.

For now, just take the defaults for the remaining options.

Click Next and then Finish in the next dialog.

The new Project will show up in the Package Explorer View.

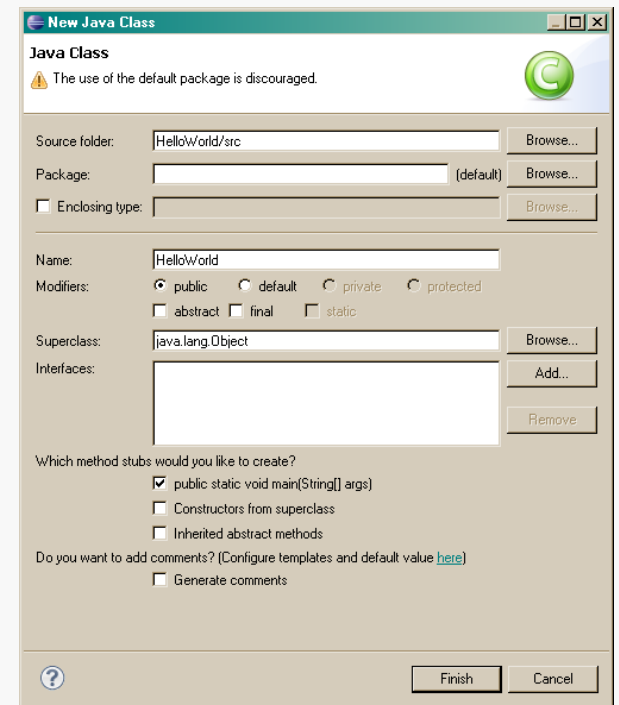


Now, select File/New/Class...

Call the class HelloWorld.

Check the box to add a public static void main() method.

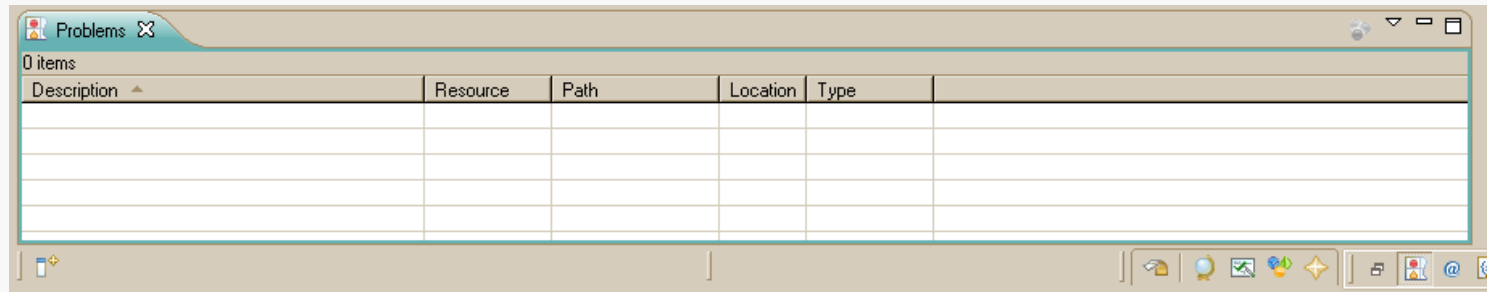
Ignore the Eclipse warning about using the default package. Click Finish.



Save the source file (not saved automatically).

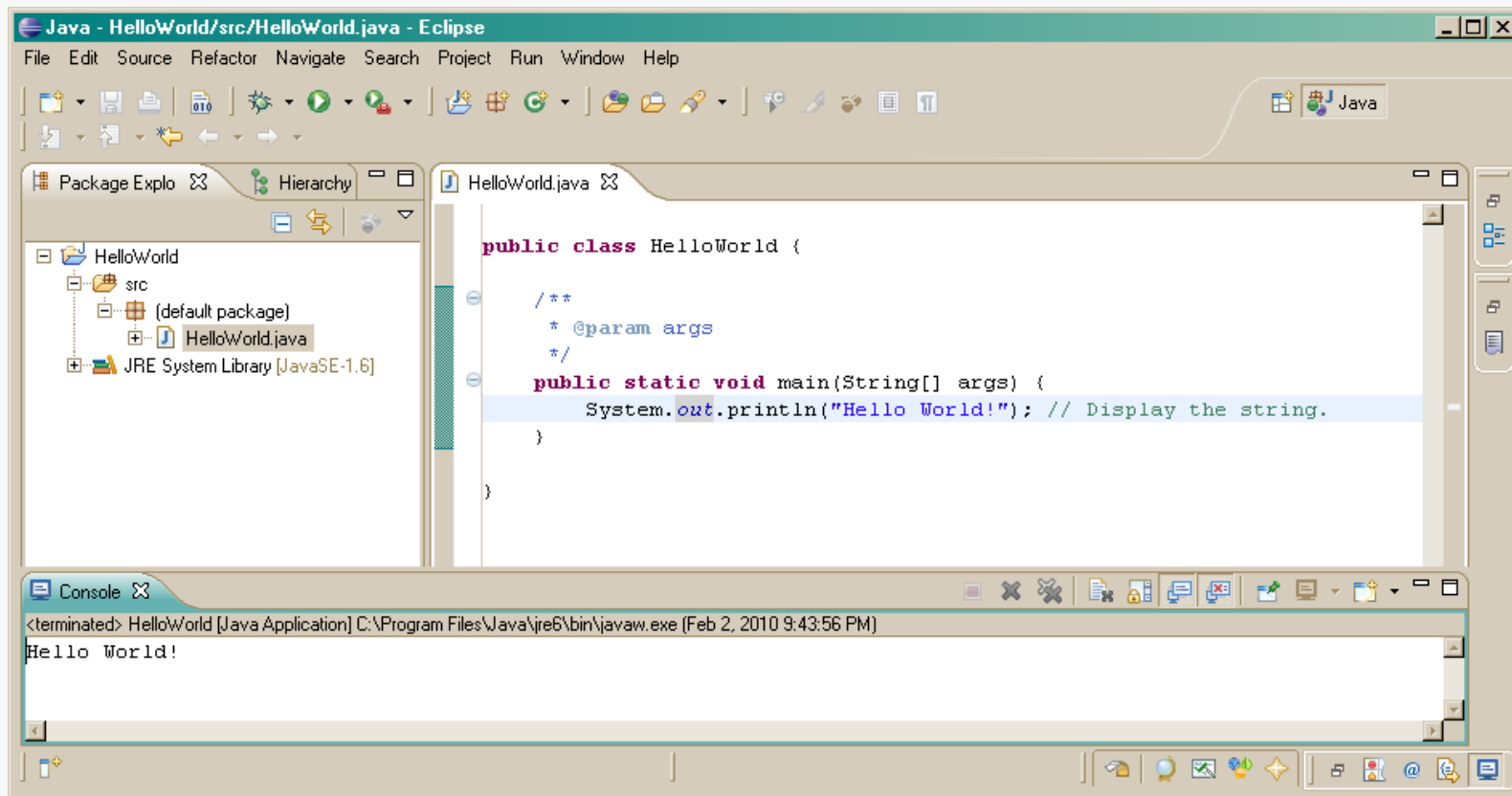
Use the Project menu or click on the Build All button ().

Any errors would be shown in the Problems View (should be none):



To execute the program, click on the Run button (▶):

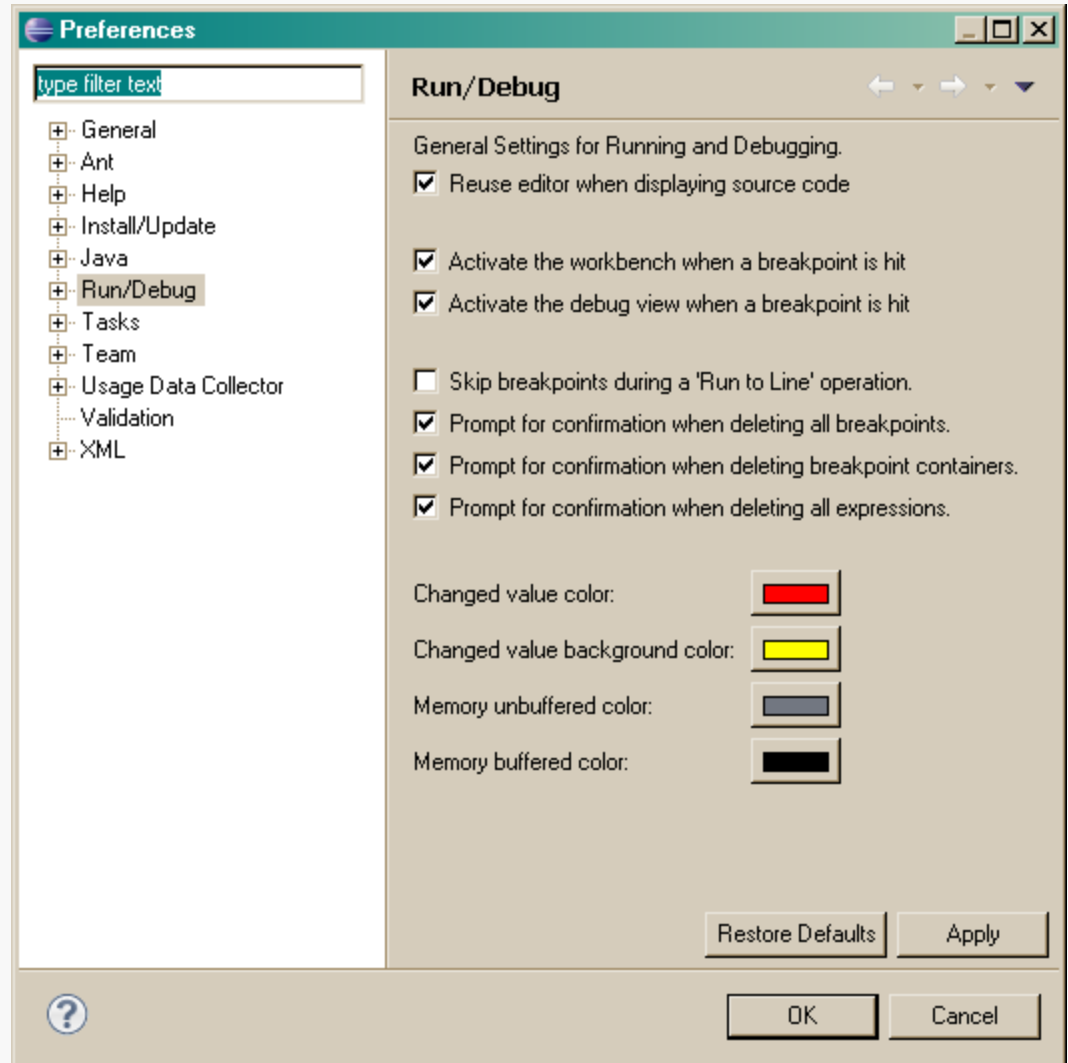
A Console View opens, showing the output to System.out:



Go to Window/Preferences to open the Preferences dialog:

There are lots of options here...

... I have a few recommendations regarding the defaults...



Under General/Workspace:

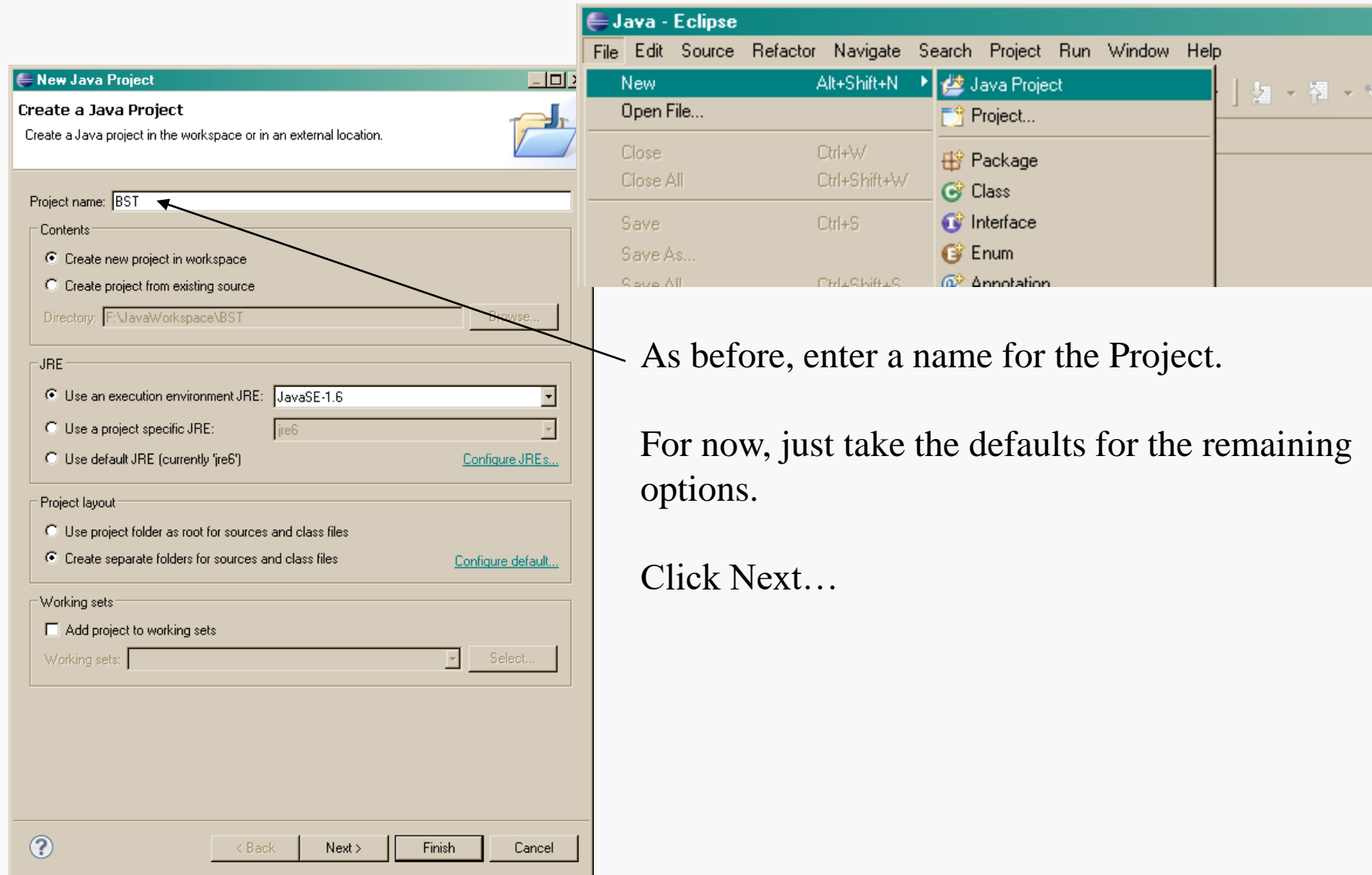
- set "Save automatically before build" so you do not have to manually save each time you want to compile your code
- unset "Build automatically"; this feature can be immensely annoying, especially on a slower machine

Under General/Text Editors:

- set the tab display width to your preference (I find 3 ideal)
- some programmers like to replace tabs with actual spaces
- set "Show line numbers"... very useful with dealing with command-line or Ant builds later on
- under Spelling: decide how much spell-checking you want
- under Keys: customize keyboard shortcuts, if you want

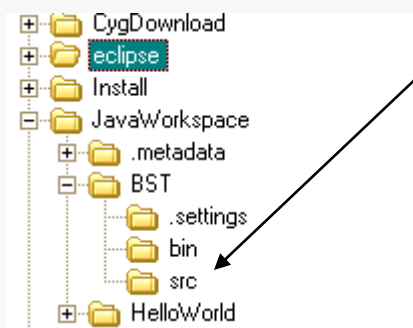
There are many other options here; some are safe to change and some are not. Explore carefully.

In the Workbench, select File/New/Java Project:

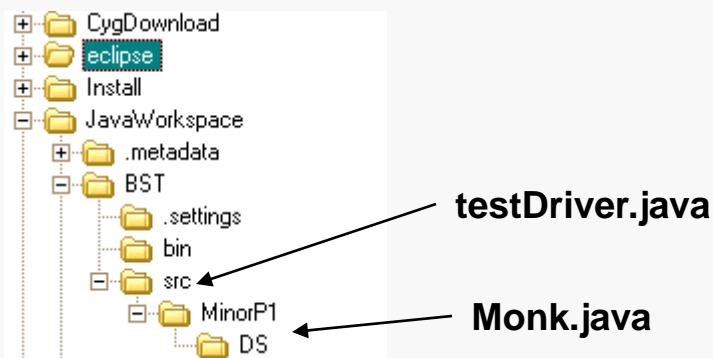


The screenshot shows the Eclipse IDE interface. On the left, the 'New Java Project' dialog is open, with the 'Project name' field containing 'BST'. An arrow points from the text 'As before, enter a name for the Project.' to this field. The dialog has sections for 'Contents', 'JRE', 'Project layout', and 'Working sets'. On the right, the 'File' menu is open, showing the path 'File > New > Java Project'. The 'New' menu item is highlighted, and the 'Java Project' option is selected. Below the dialog, there are three text annotations: 'As before, enter a name for the Project.', 'For now, just take the defaults for the remaining options.', and 'Click Next...'. The 'Next >' button is visible at the bottom of the dialog.

From the course Projects page, download the supplied zip file and unzip it in the src directory in the BST project tree:



There should now be a subtree in the directory structure:

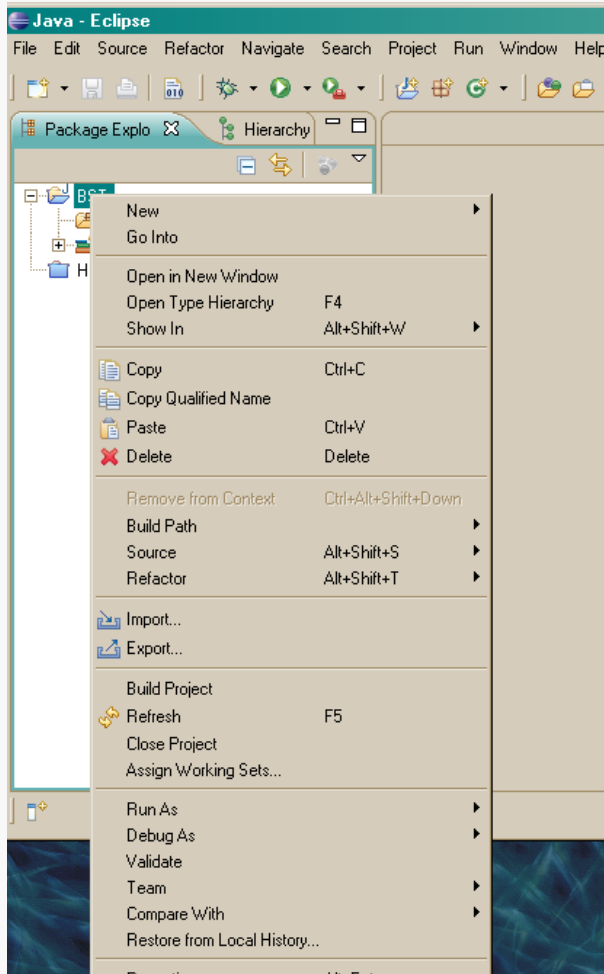


Two classes have been added, `testDriver` and `Monk`.

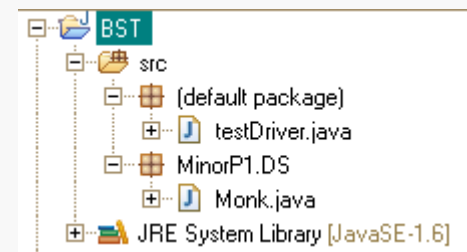
`Monk` is in a package named `MinorP1.DS` and organized in a matching directory tree.

Be sure to select "keep folders" or a similar option when unzipping the supplied file.

Right-click on the BST project icon and select Refresh on the resulting menu (or press F5):



This will cause Eclipse to recognize the added files:



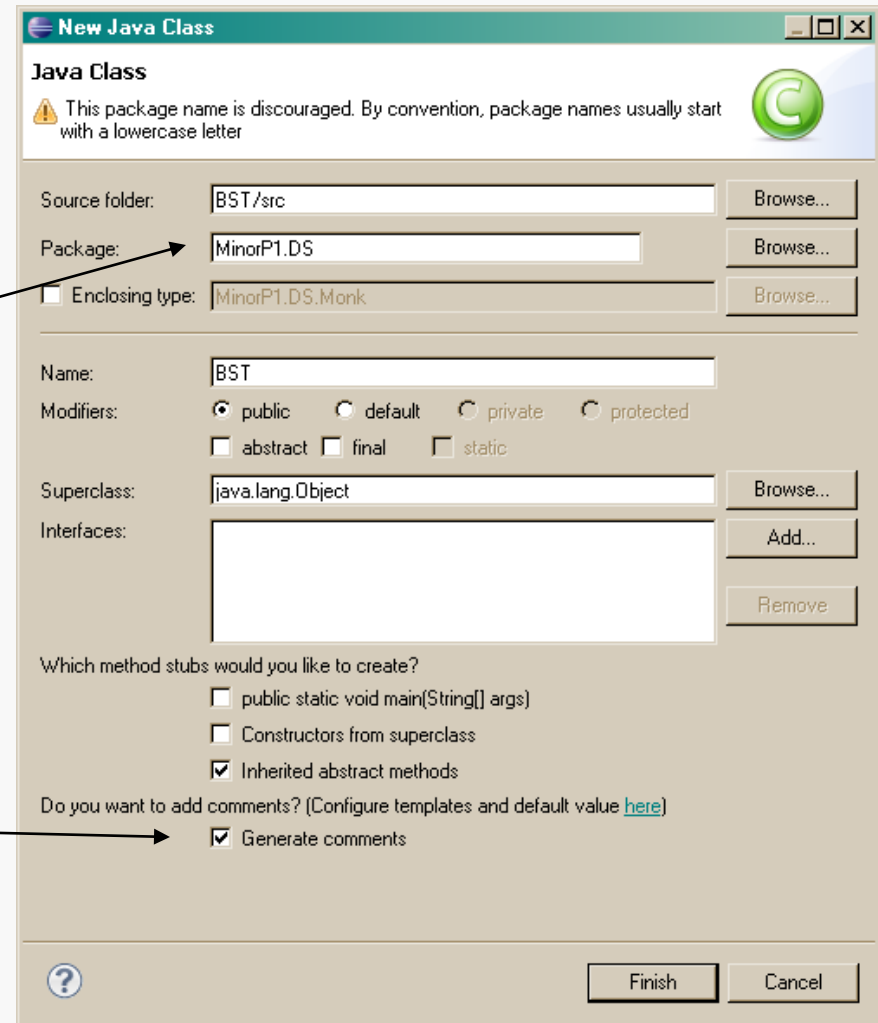
Go to File/New/Class and add the BST class:

By default, Eclipse will add the new class to the (single) package that exists in the project definition.

The source file will be in the DS subdirectory since it's in the package.

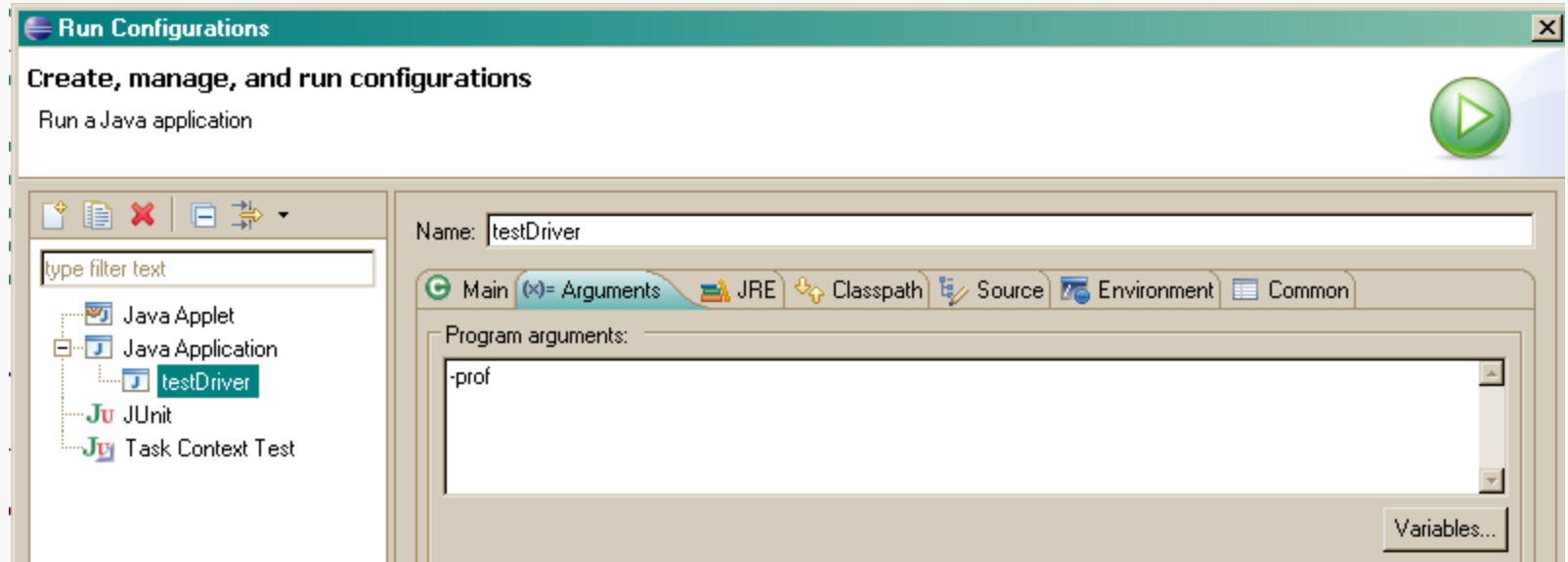
You may choose to have Eclipse generate default comment blocks if you want...

Now, just write your BST implementation and test it...



The supplied test driver needs to have a command-line argument in order to run correctly.

Go to the drop-list for the Run button  and select Run configurations...



Expand the Java Application tree, select testDriver, then select the Arguments tab...

... enter -prof as shown...

Once you have written enough code to get things to compile, click the Run button.

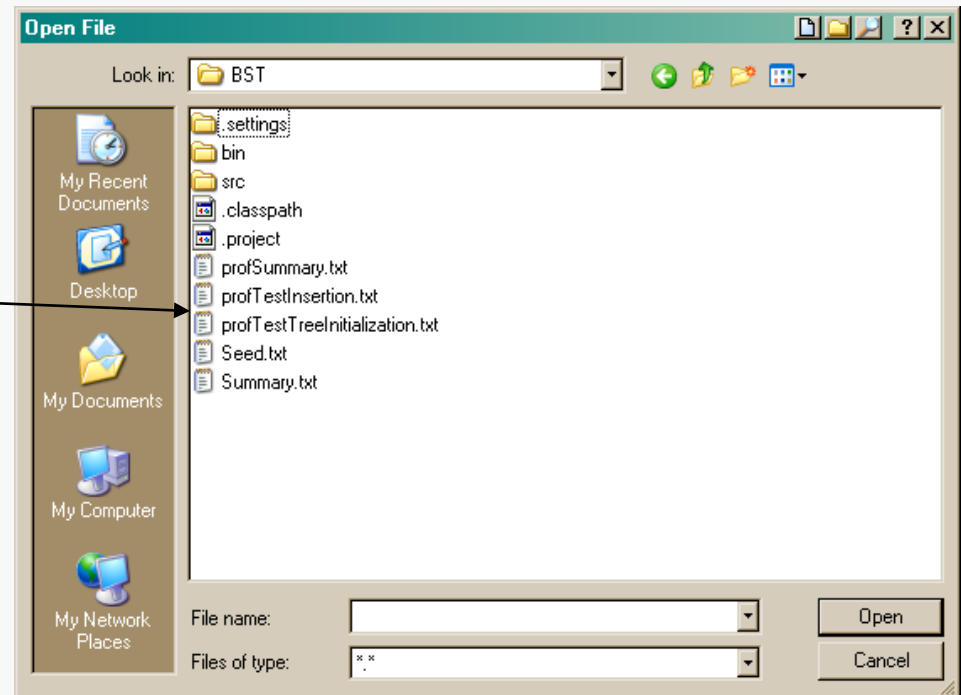
In this case, there shouldn't be any output in the Console view (see earlier slide).

Go to File/Open file and navigate to the directory where you placed the BST project:

The log files created by the test driver will be in the top-level directory for the project.

You can open them in Eclipse to check the results.

If you want to re-run the test driver with the same data as the last run, go to the Run configuration dialog and remove the command-line argument.

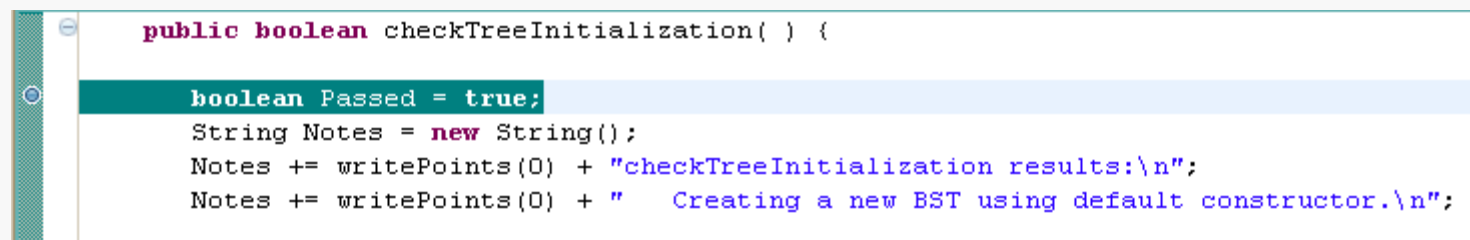


If you need to examine details of your program's execution, Eclipse provides an integrated debugger.

The first step is usually to set one or more *breakpoints*:

- pick a line of code at which you want to pause the program
- right-click in the left margin, and select Toggle Breakpoint from the menu

For example:

A screenshot of the Eclipse IDE showing a Java code editor. The code is as follows:

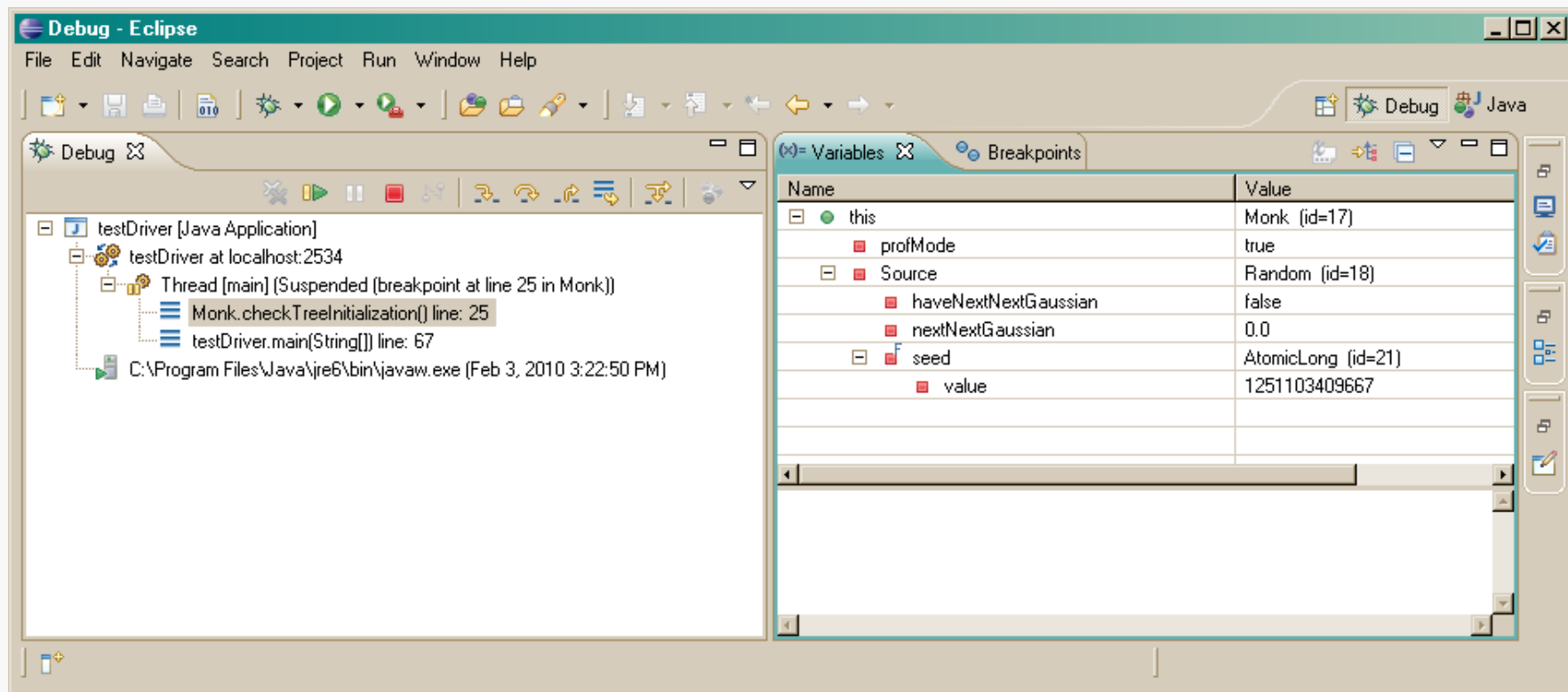
```
public boolean checkTreeInitialization( ) {  
    boolean Passed = true;  
    String Notes = new String();  
    Notes += writePoints(0) + "checkTreeInitialization results:\n";  
    Notes += writePoints(0) + "    Creating a new BST using default constructor.\n";  
}
```

The line `boolean Passed = true;` is highlighted in blue. In the left margin, a blue circle indicates that a breakpoint is set on this line.

The blue circle indicates that a breakpoint is set at that line...

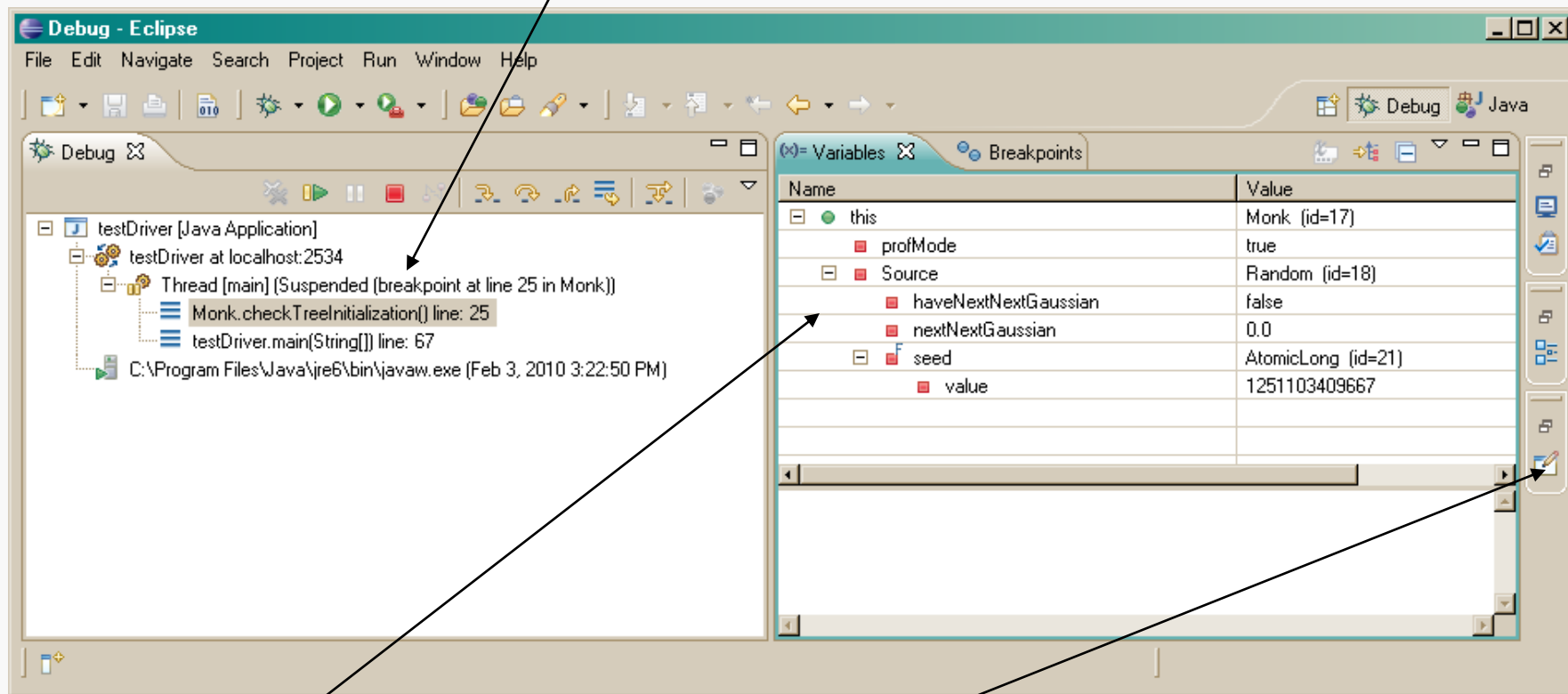
Select Run/Debug and click Yes in the following dialog...

This opens the Debug View:



You may see a different window layout; feel free to close other Views, like Outline if they are visible.

Note that your program is suspended at the breakpoint you set:



In the Variables view, you can see current values for the variables that are "active".

I find it useful to open the Editor area...

Now I want to step through the code, line by line, and check results:

The screenshot shows the Eclipse IDE in Debug mode. The top toolbar contains various debugging icons, with the Step-into button (a blue arrow pointing down) highlighted. The Debug Console shows the execution stack, with 'Monk.checkTreeInitialization() line: 25' selected. The Variables view shows the current state of variables: 'this' is 'Monk (id=17)', 'profMode' is 'true', and 'Source' is 'Random (id=18)'. The editor shows the code for 'checkTreeInitialization()' with a green highlight on the line 'boolean Passed = true;'. A blue arrow points from the Step-into button to the 'new BST<Integer>()' line in the code.

```
public boolean checkTreeInitialization() {  
    boolean Passed = true;  
    String Notes = new String();  
    Notes += writePoints(0) + "checkTreeInitialization results:\n";  
    Notes += writePoints(0) + "    Creating a new BST using default constructor.\n";  
  
    BST<Integer> Tree = new BST<Integer>();  
    Notes += writePoints(10);  
    if ( Tree.root != null ) {  
        Notes += "    Error: BST root was NOT null.\n";  
        Passed = false;  
    }  
}
```

Click the Step-into button until the highlight reaches the instantiation of a BST object...

Click the Step-over button here:

The screenshot shows the Eclipse IDE in a debug state. The 'Debug' console on the left shows the execution flow, with 'Monk.checkTreeInitialization()' at line 31 highlighted. The 'Variables' view on the right shows the current state of variables:

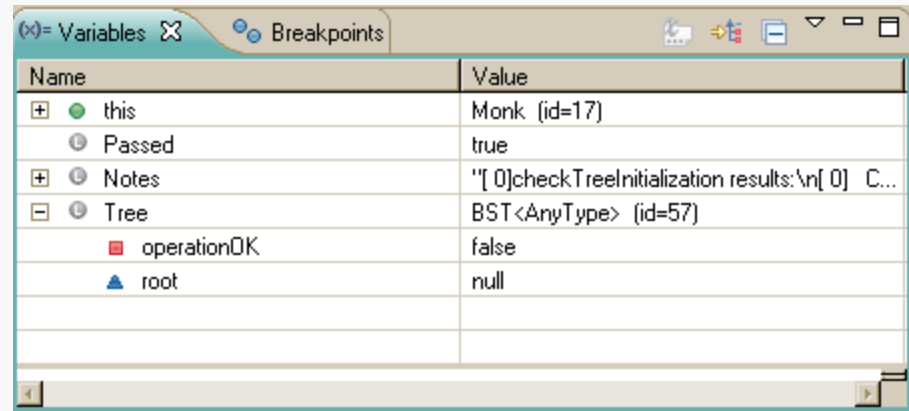
Name	Value
this	Monk (id=17)
Passed	true
Notes	"[0]checkTreeInitialization results:\n[0] C...
Tree	BST<AnyType> (id=57)

The source code editor shows the implementation of 'checkTreeInitialization()' with line 31 highlighted:

```
public boolean checkTreeInitialization() {  
    boolean Passed = true;  
    String Notes = new String();  
    Notes += writePoints(0) + "checkTreeInitialization results:\n";  
    Notes += writePoints(0) + "    Creating a new BST using default constructor.\n";  
  
    BST<Integer> Tree = new BST<Integer>();  
    Notes += writePoints(10);  
    if ( Tree.root != null ) {  
        Notes += "    Error:  BST root was NOT null.\n";  
        Passed = false;  
    }  
}
```


Check the Variables view... you should see an entry for Tree...

Expand the tree for Tree:



This shows the data members of the object Tree (not too exciting just yet)...

You can use this approach to watch your program run in enormous detail, which may reveal that it's not doing what you intended...

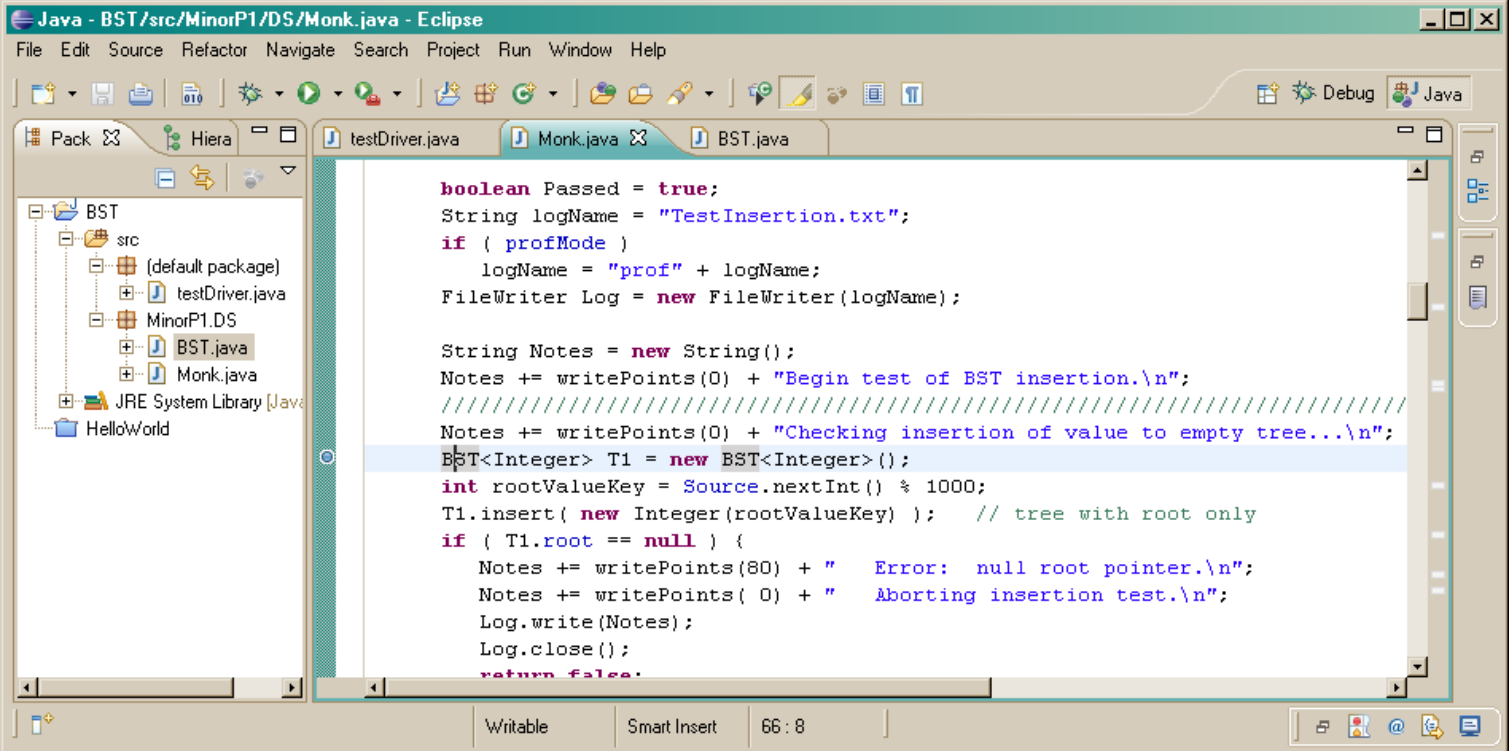
When you're ready to exit the Debug View, you can terminate your program by using the stop button () and then click the Java button in the upper-right corner to switch back to development mode.

The difference is that if you are executing a method call (or invoking `new`, for example) in the current statement:

step-into	takes you into the implementation of that method
step-over	calls the method, but does not step you through its execution

Both are useful... step-into is frustrating when system code is involved.

Remove the earlier break-point and set one at the statement inside the `Monk` method `testInsertion()` that instantiates a BST named `T1`:



```
boolean Passed = true;
String logName = "TestInsertion.txt";
if ( profMode )
    logName = "prof" + logName;
FileWriter Log = new FileWriter(logName);

String Notes = new String();
Notes += writePoints(0) + "Begin test of BST insertion.\n";
////////////////////////////////////
Notes += writePoints(0) + "Checking insertion of value to empty tree.\n";
BST<Integer> T1 = new BST<Integer>();
int rootValueKey = Source.nextInt() % 1000;
T1.insert( new Integer(rootValueKey) ); // tree with root only
if ( T1.root == null ) {
    Notes += writePoints(80) + " Error: null root pointer.\n";
    Notes += writePoints( 0) + " Aborting insertion test.\n";
    Log.write(Notes);
    Log.close();
    return false;
}
```

Run this in debug mode...

Use step-over twice to construct the empty tree and to generate a random value to insert into it; then use step-into and see what your `insert()` method does...

... stepping along until the `insert()` method returns:

The Variables View now shows the structure of the BST `T1` after the insertion of the first value:

BTW, if you expand the tree for `this`, you may get an interesting experience... I leave it to you to decide if Eclipse is being sensible...

Name	Value	
+	this	Monk (id=17)
	Passed	true
+	logName	"profTestInsertion.txt" (id=18)
+	Log	FileWriter (id=23)
+	Notes	"[0]Begin test of BST insertion.\n[0]Check..."
-	T1	BST<AnyType> (id=31)
	operationOK	true
	- root	BST\$BinaryNode (id=54)
	- element	Integer (id=33)
	value	394
	- left	null
	- right	null
	+ this\$0	BST<AnyType> (id=31)
	rootValueKey	394

Set another breakpoint at the following line of code in Monk:

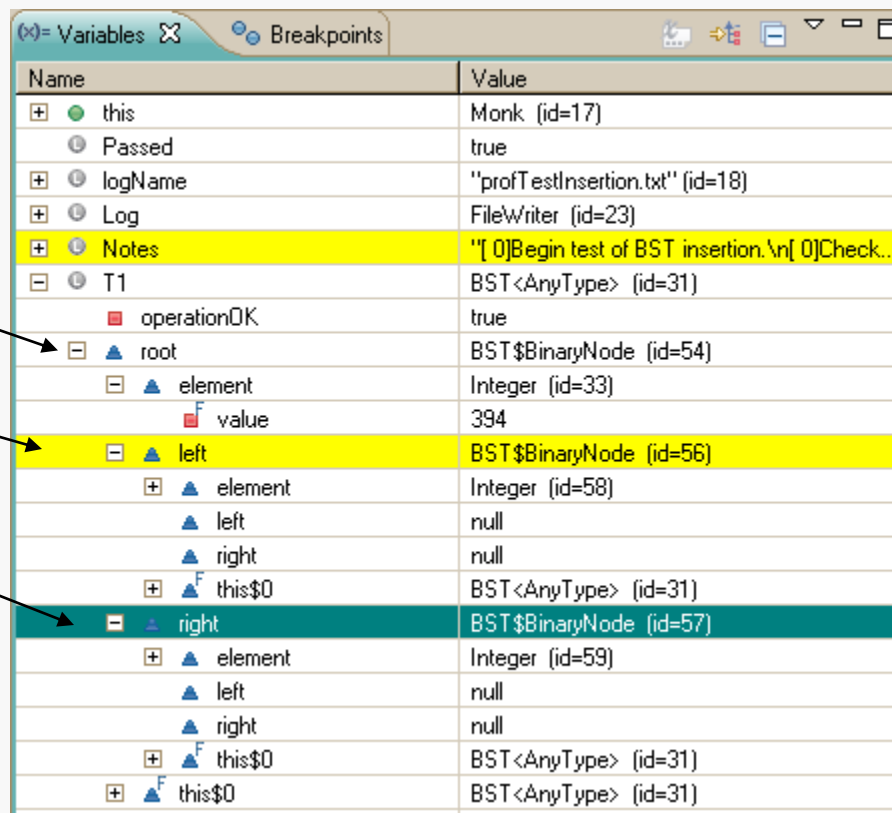
```
BST<Integer>.BinaryNode rootAddress = T1.root;
```

Click the Resume () button to execute up to the new breakpoint, and look at T1 now:

There's T1.root...

and the left child...

and the right child...



Name	Value	
+	this	Monk (id=17)
○	Passed	true
+	logName	"profTestInsertion.txt" (id=18)
+	Log	FileWriter (id=23)
+	Notes	"[0]Begin test of BST insertion.\n[0]Check...
[-]	T1	BST<AnyType> (id=31)
	operationOK	true
[-]	root	BST\$BinaryNode (id=54)
[-]	element	Integer (id=33)
	value	394
[-]	left	BST\$BinaryNode (id=56)
+	element	Integer (id=58)
	left	null
	right	null
+	this\$0	BST<AnyType> (id=31)
[-]	right	BST\$BinaryNode (id=57)
+	element	Integer (id=59)
	left	null
	right	null
+	this\$0	BST<AnyType> (id=31)
+	this\$0	BST<AnyType> (id=31)