Eclipse

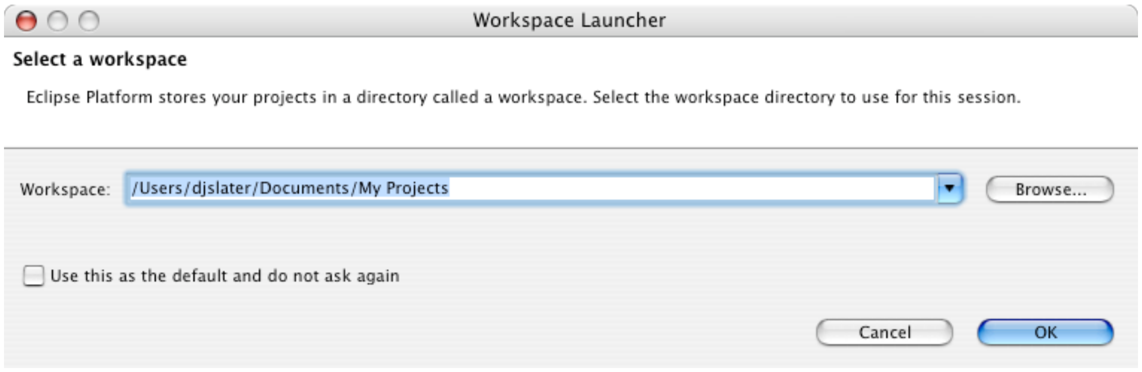
Eclipse is a free open-source Java environment available from http://eclipse.org. Eclipse is a Java program, but it uses a custom user interface toolkit that does not run on all platforms that supports Java 2. Check the web site for supported platforms. Eclipse requires a Java 2 runtime, so you need to install the Java 2 SDK first before installing Eclipse. You can download the SDK from http://java.sun.com/j2se/. *For this course, you will want the J2SE 1.4.2 release. (If you have a Macintosh, the Java 2 SDK is already installed)*

Starting Eclipse

When you start Eclipse, a startup screen appears, and the program spends some time loading various modules.

eclipse
3.1
THE ECLIPSE PROJECT
(c) Copyright Eclipse contributors and others. 2000, 2004. All rights reserved. Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S., other countries, or both.

Eclipse will ask you for your workspace (where on your computer the projects you are working on will be stored.) The prompt will look like the following: Click on the **Browse...** button to navigate to the directory of your choice if it is not already selected or in the drop-down menu...


Workspace Launcher

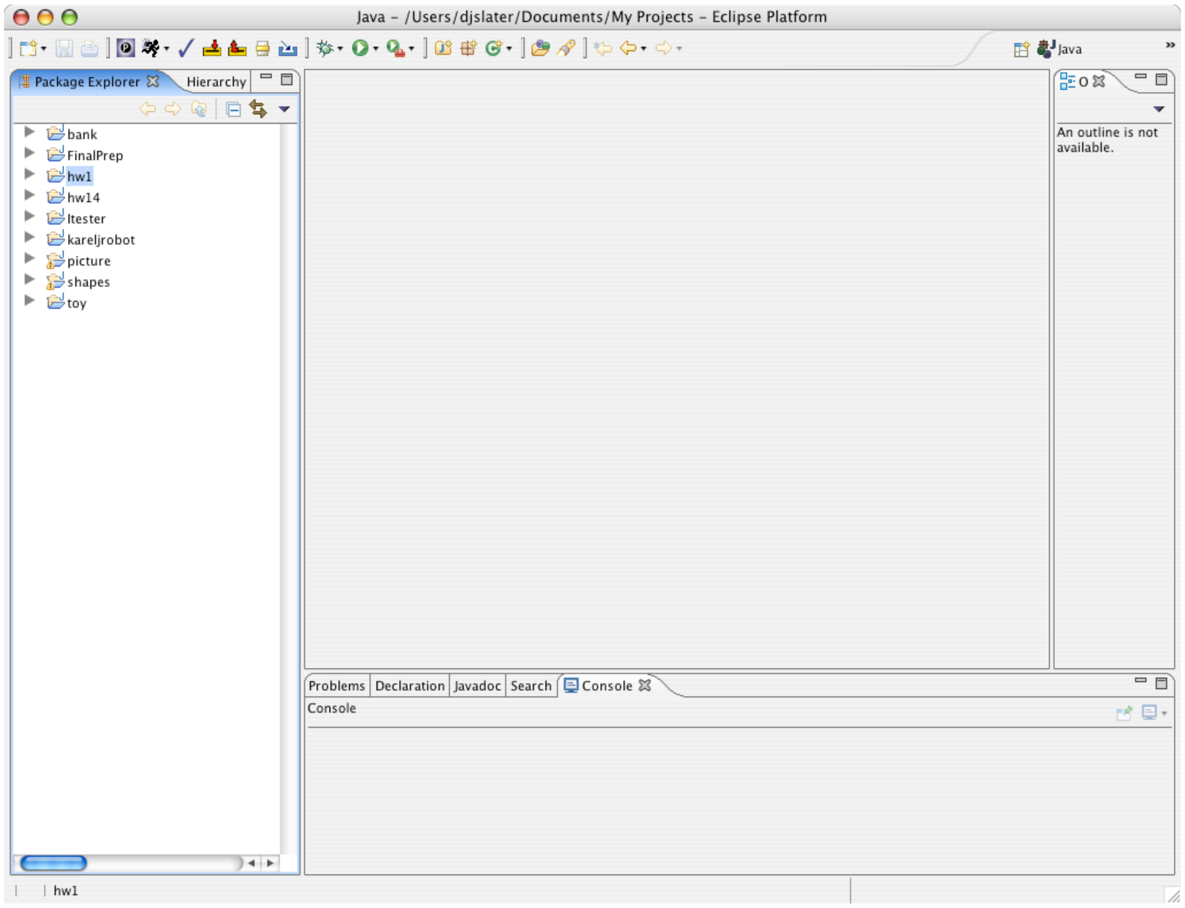
Select a workspace

Eclipse Platform stores your projects in a directory called a workspace. Select the workspace directory to use for this session.

Workspace: /Users/djslater/Documents/My Projects

☐ Use this as the default and do not ask again

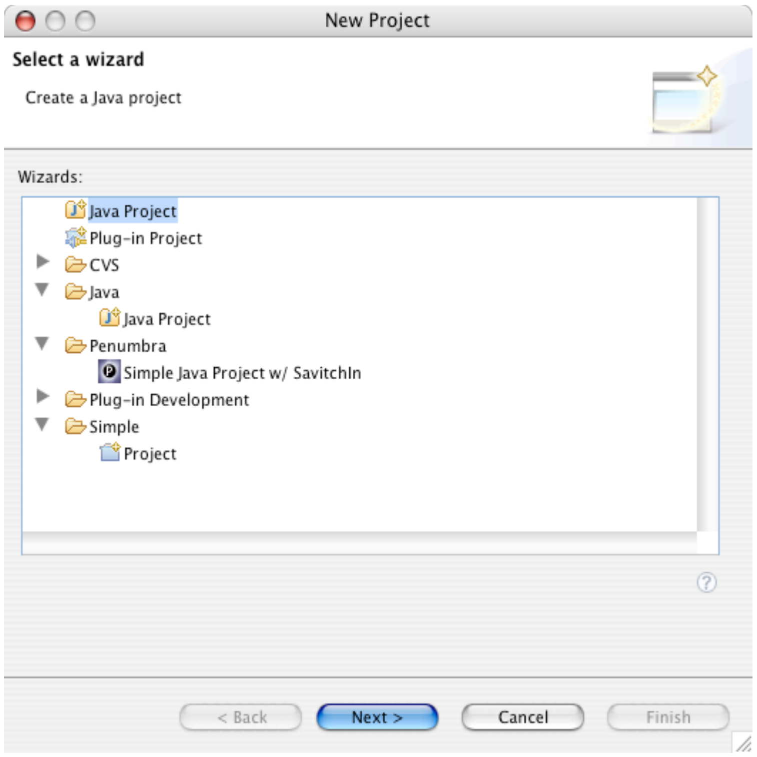
When Eclipse has finished loading, you see a screen similar to the following:



Loading an existing program

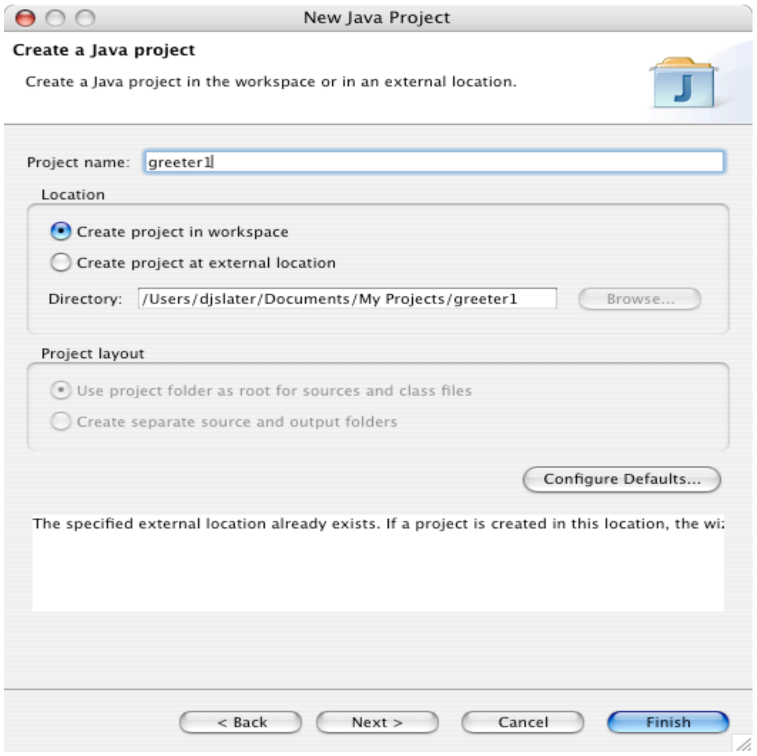
If you already have your program in project *workspace* (see above), but it is not in the **Package Hierarchy** window on the left of the screen, then you need to make a *project* that contains the file(s). Follow these steps.

Select **File->New->Project** from the menu. You will get the following dialog.



Select the **Java Project** option and click on the **Next>** button. (*Do not select the Simple option!)*

In the following dialog, give a name to the project, the same name as the directory that contains the files (watch spelling and capitalization). There should be a message indicating *The specified external location already exists*. Eclipse will then attempt to import all the resources available at that location. Check **Create project in workspace** if it has not been selected.



Project name: greeter1

Location
- [✓ Create project in workspace]
- Create project at external location

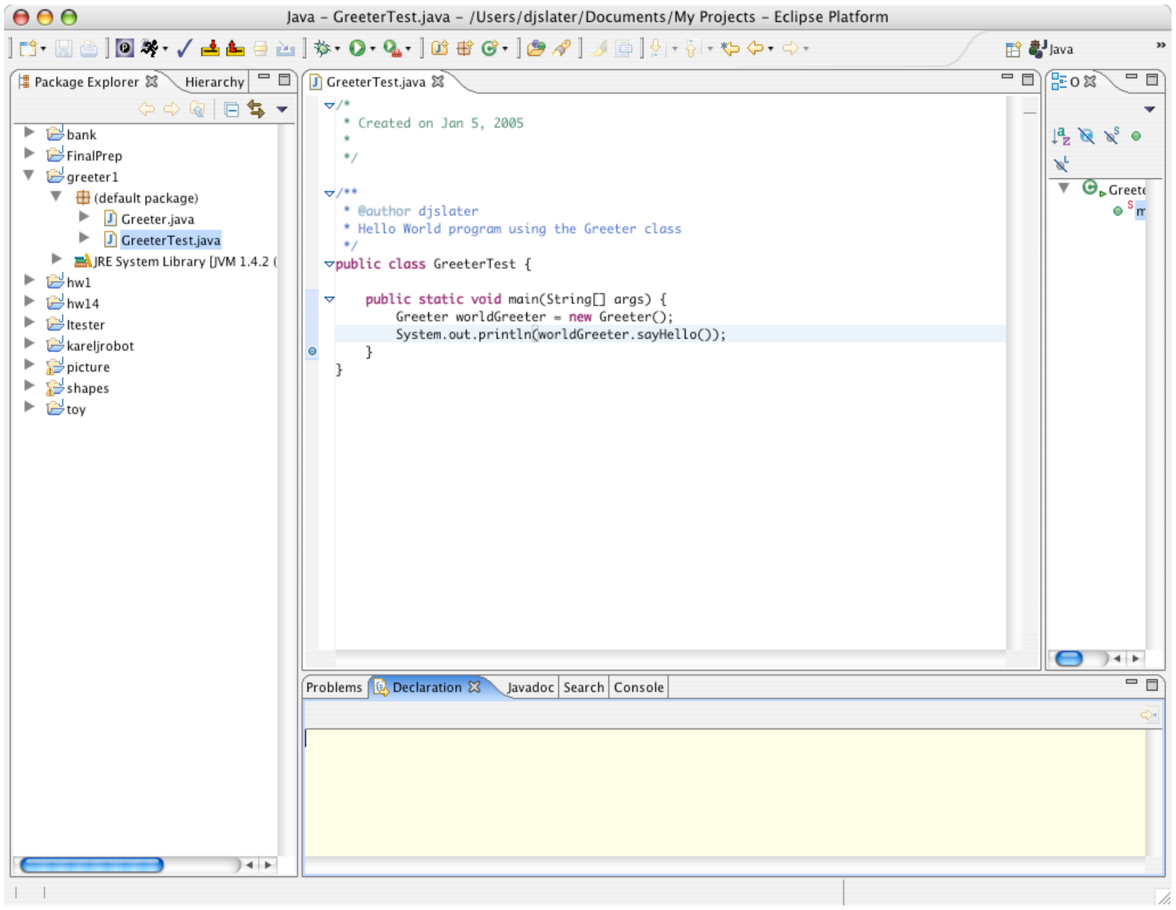
Directory: /Users/djslater/Documents/My Projects/greeter1 Browse...

Project layout
- [✓ Use project folder as root for sources and class files]
- Create separate source and output folders

The specified external location already exists. If a project is created in this location, the wi:

Click on the **Finish** button.

The project appears in the left hand panel. Expand it, and also expand the **default package** icon. Double-click on one of the file names. The file is displayed in an edit window:



```
/*
 * Created on Jan 5, 2005
 *
 */

/**
 * @author djslater
 * Hello World program using the Greeter class
 */
public class GreeterTest {

    public static void main(String[] args) {
        Greeter worldGreeter = new Greeter();
        System.out.println(worldGreeter.sayHello());
    }
}
```

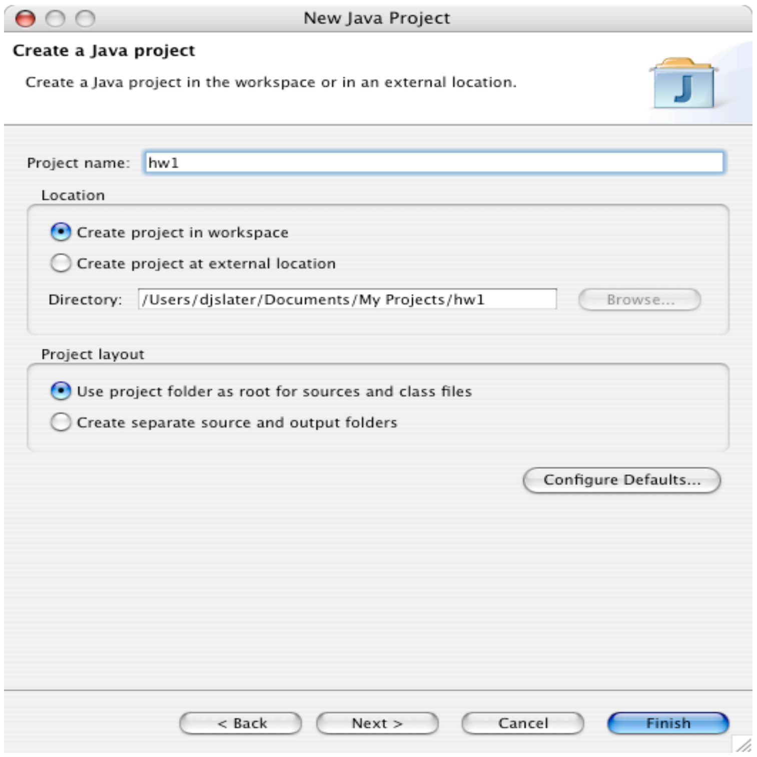
Starting a new program

If you write a program from scratch, then you can start your work in Eclipse. It is always best to place each of your programs into a separate directory. Eclipse will create the directory for you.

Select **File->New->Project** from the menu. You will get the **New Project** dialog.

Select the **Java** option and click on the **Next>** button. (*Do not select the Simple option!)*

In the following dialog, give a name to the project. Accept the *default* choices, **Create project in workspace** and **Use project folder as root for sources and class files**



Create a Java project

Create a Java project in the workspace or in an external location.

Project name: hw1

Location

(●) Create project in workspace

( ) Create project at external location

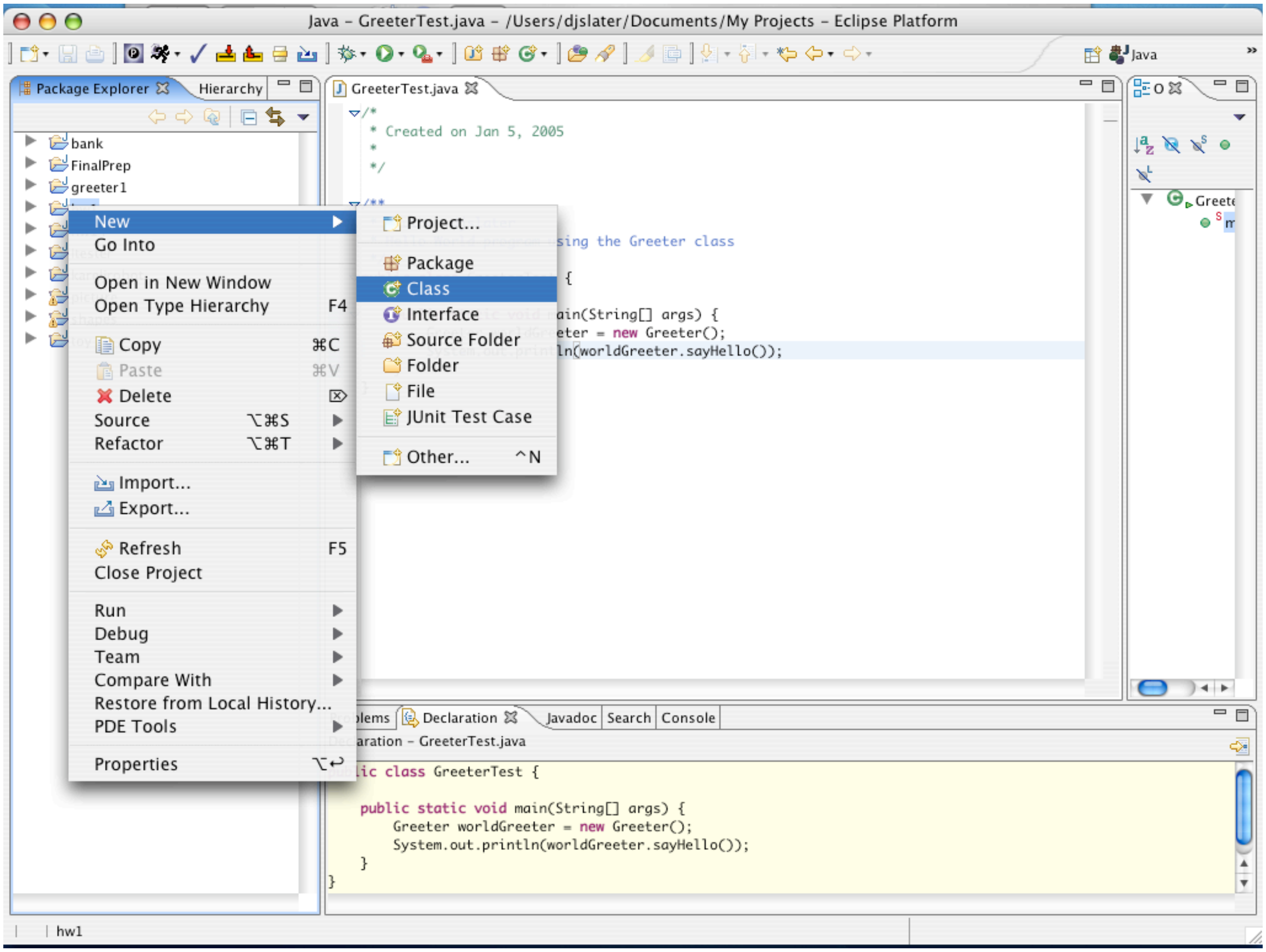
Directory: /Users/djslater/Documents/My Projects/hw1

Project layout

(●) Use project folder as root for sources and class files

( ) Create separate source and output folders

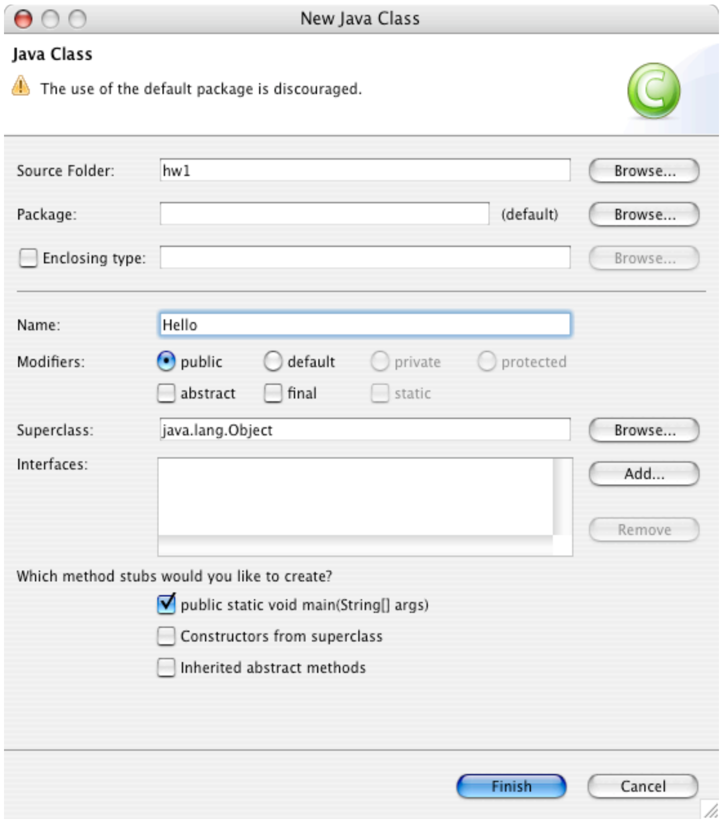
Click on the **Finish** button. Now locate the name of your new project in the left hand panel. Click on it with the *right* mouse button (Windows) or *control-click* (Macintosh). Select **New->Class** from the context menu
.



```
public class GreeterTest {

    public static void main(String[] args) {
        Greeter worldGreeter = new Greeter();
        System.out.println(worldGreeter.sayHello());
    }
}
```

The **New Class** dialog appears.

Supply the name of the class. If you want a `main` method for this class, check the box **public static void main(String[] args)**.



Java Class

The use of the default package is discouraged.

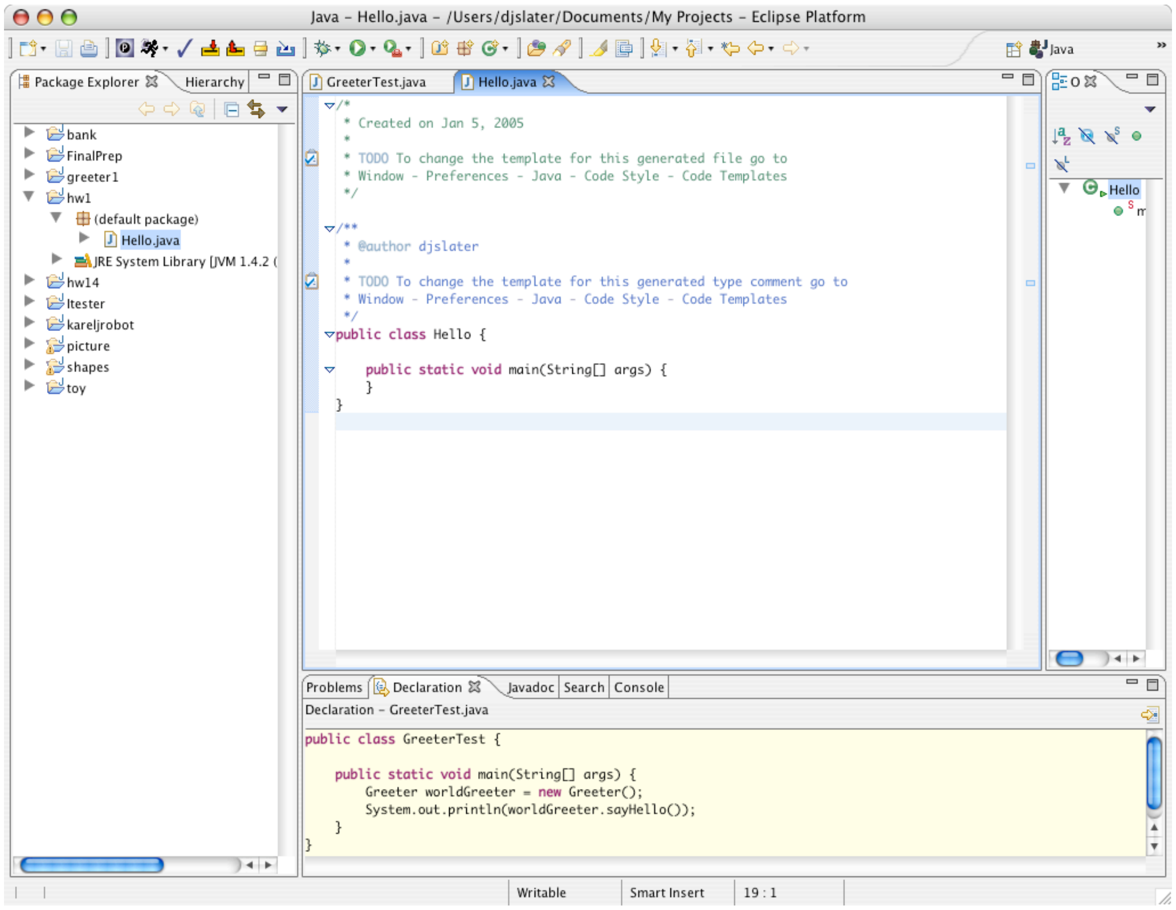
Source Folder:	hw1
Package:	(default)
Enclosing type:	☐
Name:	Hello
Modifiers:	[✓ public] default private protected; ☐ abstract ☐ final ☐ static
Superclass:	java.lang.Object
Interfaces:	

Which method stubs would you like to create?

- ☑ public static void main(String[] args)
- ☐ Constructors from superclass
- ☐ Inherited abstract methods

Click on the **Finish** button.

Finally, you get an editor window into which you can type your program.

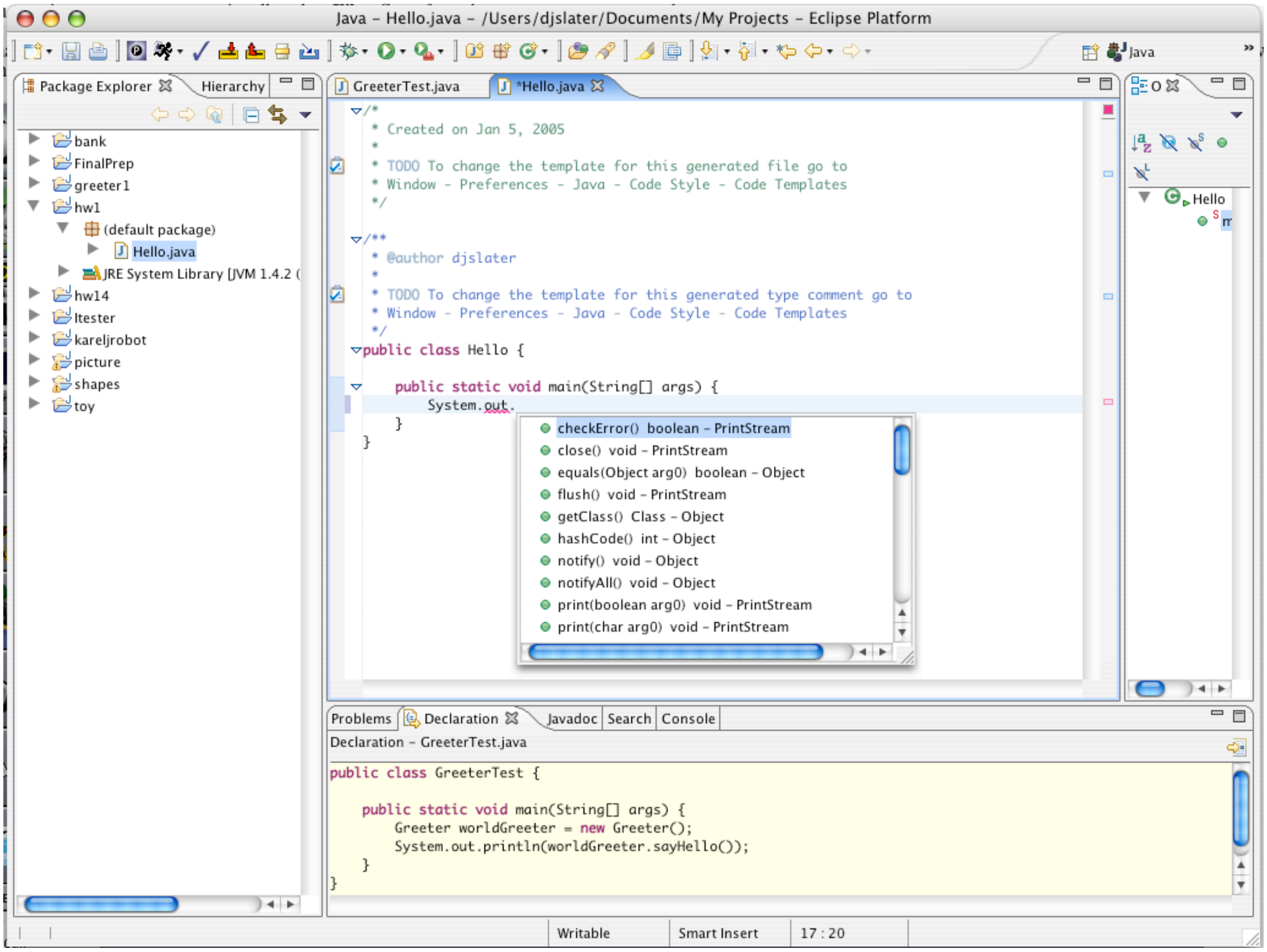


```
/*
 * Created on Jan 5, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author djslater
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Hello {

    public static void main(String[] args) {
    }
}
```

```
public class GreeterTest {

    public static void main(String[] args) {
        Greeter worldGreeter = new Greeter();
        System.out.println(worldGreeter.sayHello());
    }
}
```

As you type in your program, occasionally select **File->Save** from the menu to save your work.

You may enjoy the "content assist" feature of Eclipse. If you type a partial input and then hit CTRL+SPACE, a dialog shows all possible completions. Just pick the one you want from the list.



```
/*
 * Created on Jan 5, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author djslater
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Hello {

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

```
checkError()  boolean - PrintStream
close()  void - PrintStream
equals(Object arg0)  boolean - Object
flush()  void - PrintStream
getClass()  Class - Object
hashCode()  int - Object
notify()  void - Object
notifyAll()  void - Object
print(boolean arg0)  void - PrintStream
print(char arg0)  void - PrintStream
```

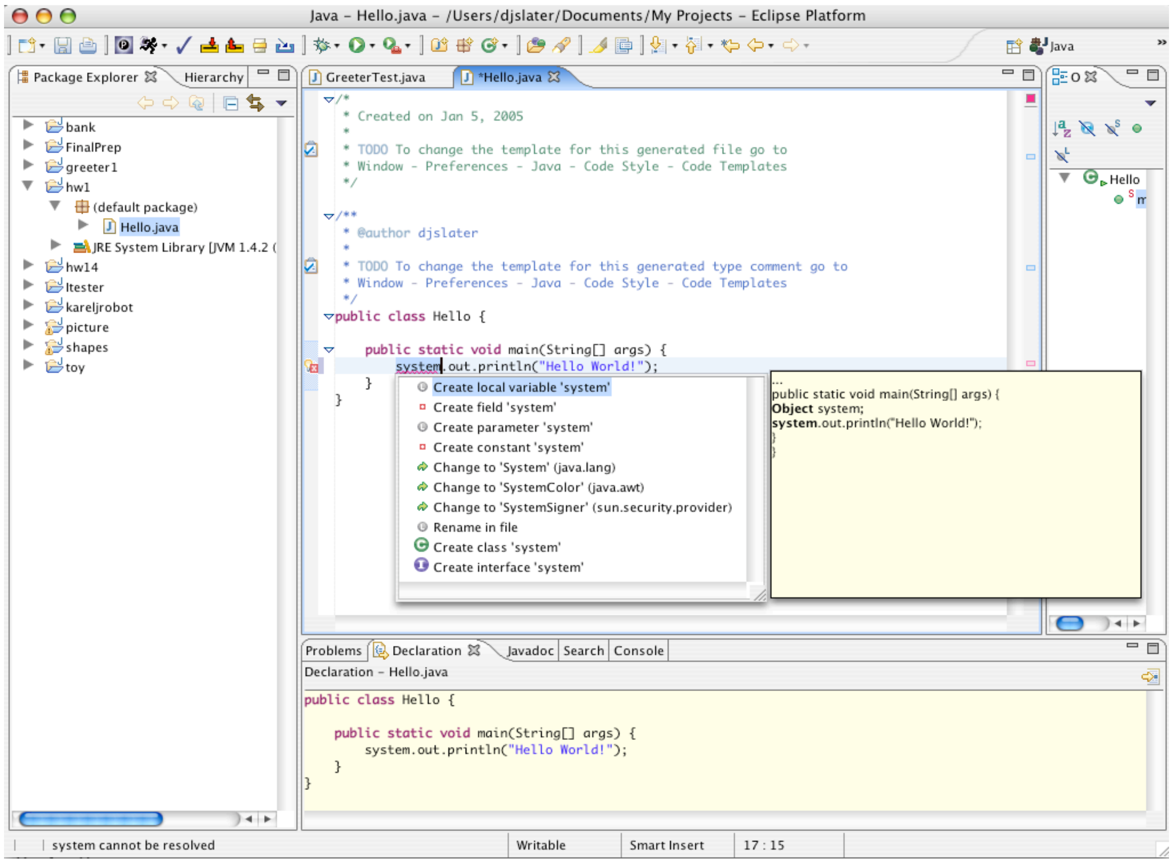
```
public class GreeterTest {

    public static void main(String[] args) {
        Greeter worldGreeter = new Greeter();
        System.out.println(worldGreeter.sayHello());
    }
}
```

You may also want to experiment with some of the other convenience features of Eclipse, such as **Source->Generate getter and setter**.

Finding Errors

Even before compiling a program in Eclipse, the Editor can display problems in your program. (*See the red X in the leftmost column in the editor window below.*)

By double-clicking on the red X, a description of the problem and some helpful options are displayed. *In this case, you would click on* **Change to 'System' (java.lang)**


Java - Hello.java - /Users/djslater/Documents/My Projects - Eclipse Platform
Package Explorer
Hierarchy
bank
FinalPrep
greeter1
hw1
(default package)
Hello.java
JRE System Library [JVM 1.4.2 (
hw14
ltester
kareljrobot
picture
shapes
toy
GreeterTest.java
*Hello.java
Hello
Create local variable 'system'
Create field 'system'
Create parameter 'system'
Create constant 'system'
Change to 'System' (java.lang)
Change to 'SystemColor' (java.awt)
Change to 'SystemSigner' (sun.security.provider)
Rename in file
Create class 'system'
Create interface 'system'
Problems
Declaration
Javadoc
Search
Console
Declaration - Hello.java
system cannot be resolved
Writable
Smart Insert
17 : 15

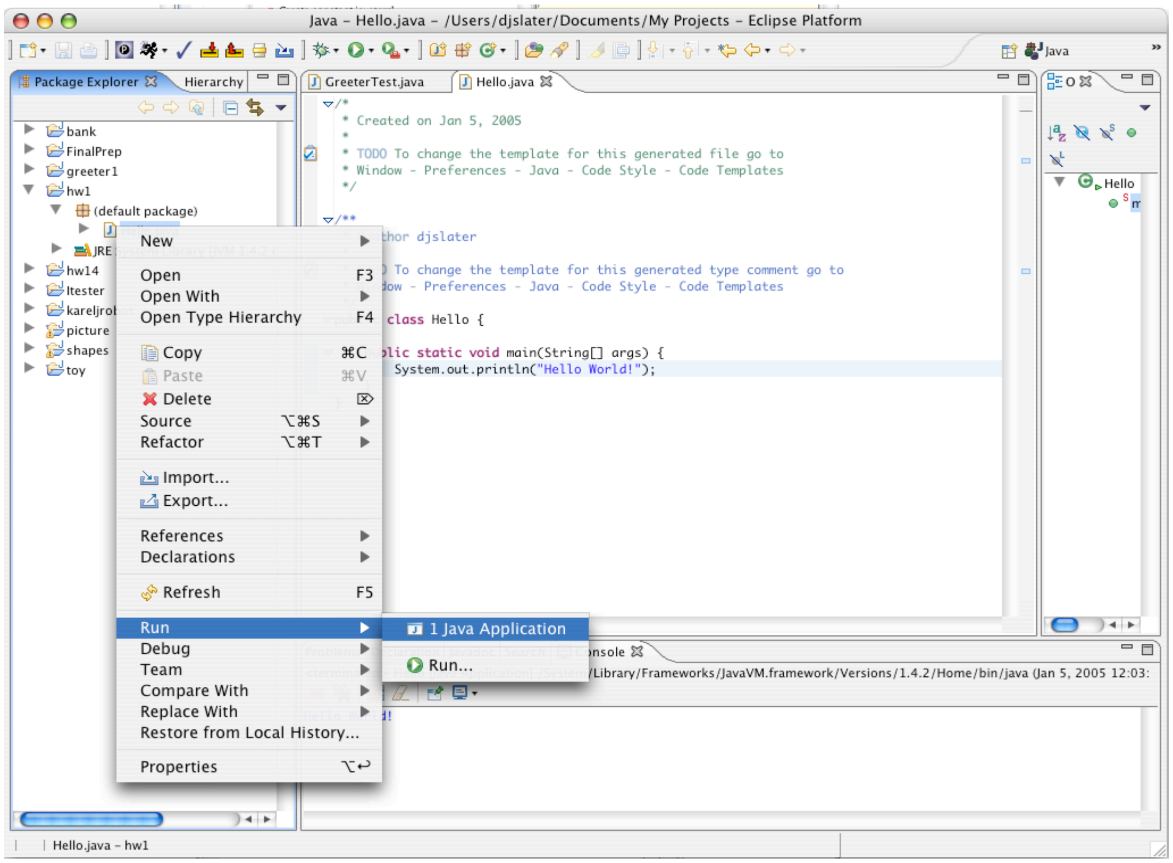
```
/*
 * Created on Jan 5, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author djslater
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Hello {

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

```
...
public static void main(String[] args) {
Object system;
system.out.println("Hello World!");
}
}
```

```
public class Hello {

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

Running a program

To run a program, *right-click* (Windows) or *control-click* (Macintosh) on the file with the ***main*** method, and from the context menu select the **Run->Java Application** menu option. The program runs. Any console output is directed to a window at the bottom of the screen.

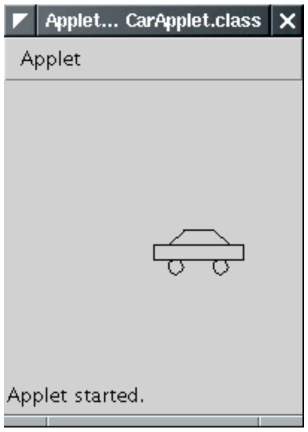


```
/*
 * Created on Jan 5, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 ...thor djslater
 ... To change the template for this generated type comment go to
 ...dow - Preferences - Java - Code Style - Code Templates
 ... class Hello {
 ...lic static void main(String[] args) {
      System.out.println("Hello World!");
```

```
/Library/Frameworks/JavaVM.framework/Versions/1.4.2/Home/bin/java (Jan 5, 2005 12:03:
```

Running applets

To run an applet, make sure the current project is still selected in the leftmost pane. Then select the menu option **Run->Java Applet**. Eclipse will launch the applet viewer on the subclass of `Applet` that it finds in your project.

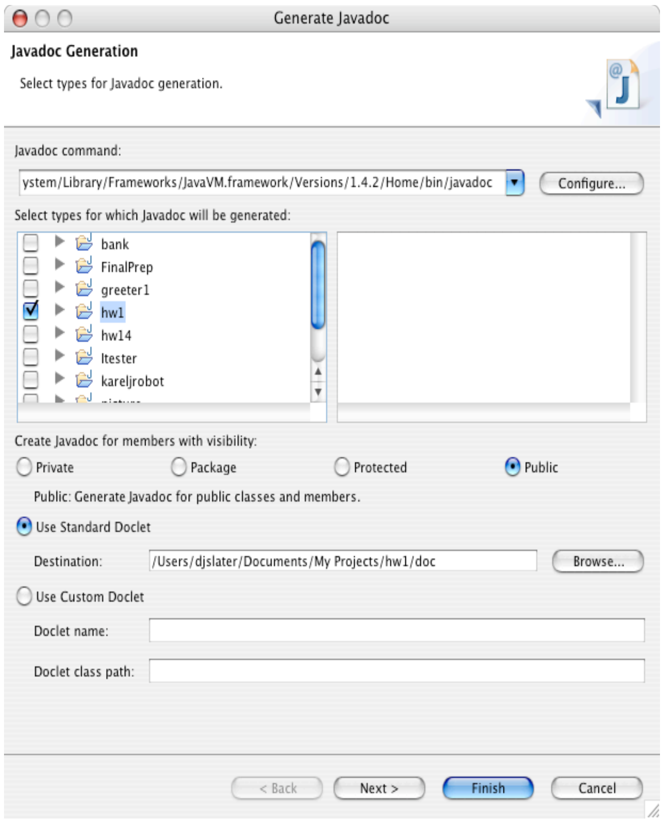


The applet viewer is started with a default size which is rather small. Just resize the applet window.

Close the window to terminate the applet.

Generating Javadoc comments

Select **Project -> Generate Javadoc** from the menu. You will get a dialog such as the following one:



Javadoc Generation

Select types for Javadoc generation.

Javadoc command: ystem/Library/Frameworks/JavaVM.framework/Versions/1.4.2/Home/bin/javadoc

Select types for which Javadoc will be generated:

- [ ] bank
- [ ] FinalPrep
- [ ] greeter1
- [x] hw1
- [ ] hw14
- [ ] Itester
- [ ] kareljrobot

Create Javadoc for members with visibility:

Private Package Protected [✓ Public]

Public: Generate Javadoc for public classes and members.

[✓ Use Standard Doclet]

Destination: /Users/djslater/Documents/My Projects/hw1/doc

Use Custom Doclet

Doclet name:

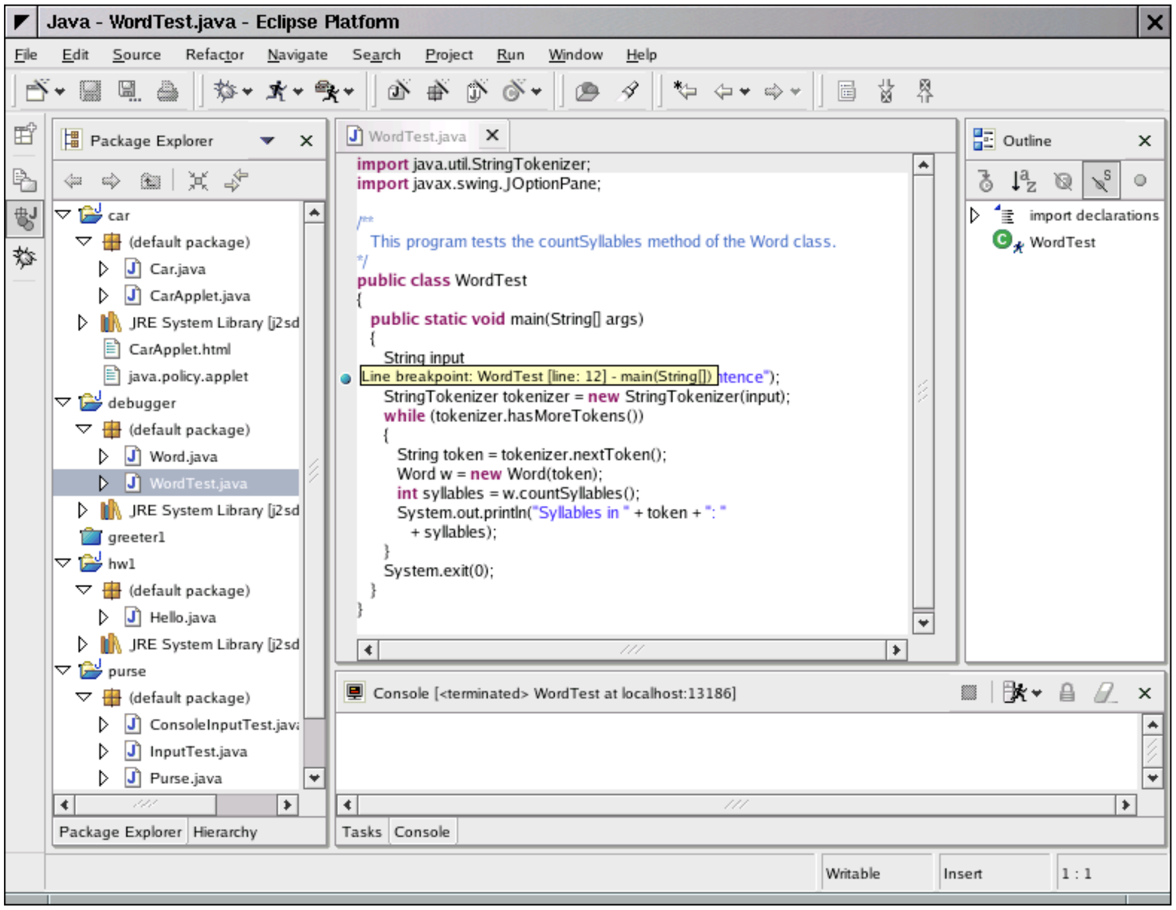
Doclet class path:

Click the cleckbox for the project that you want to document. Then make sure that the destination directory is correct. Click on **Finish**.

Tracing through a program.

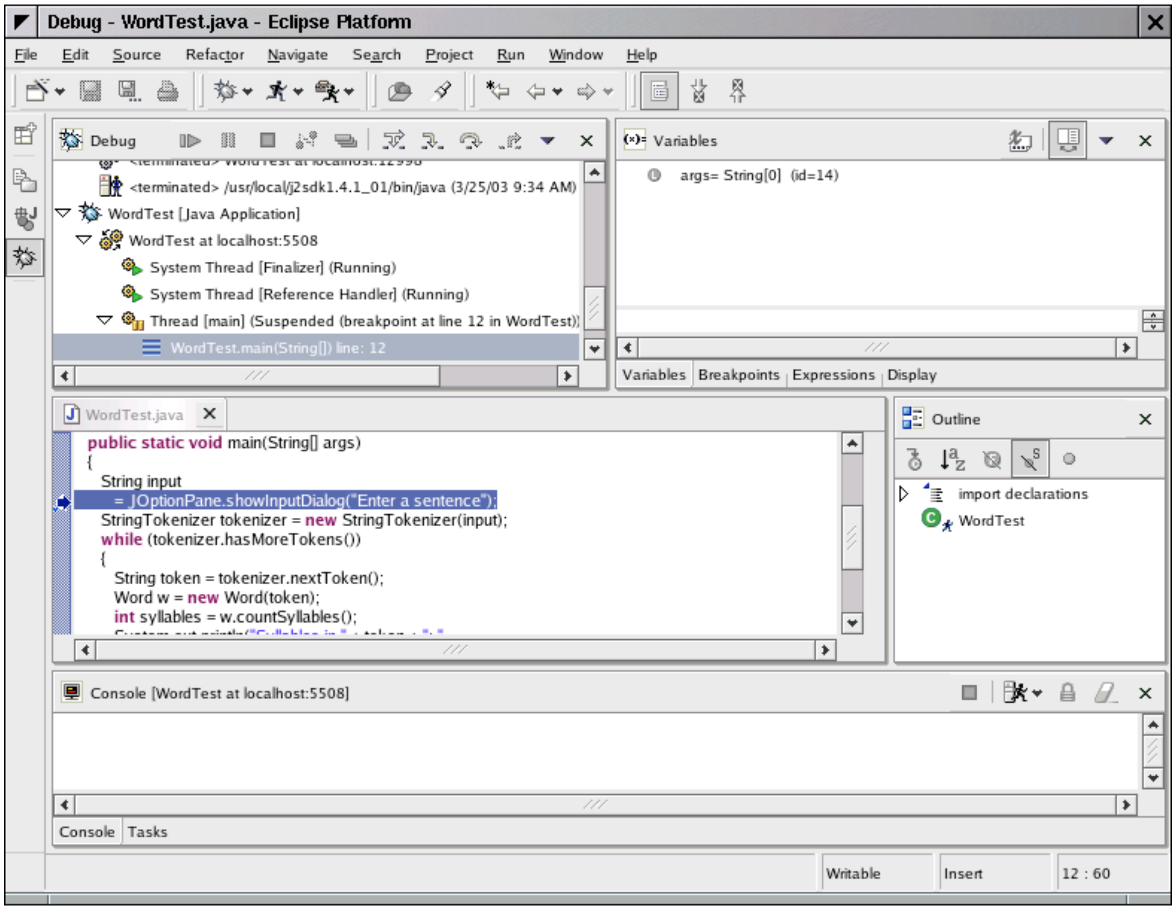
Before debugging a program, you should set a breakpoint at the start of the `main` method.

Double-click on the gray bar to the left of the edit window, next to the first source line *after* the line `public static void main(String[] args)`. A blue dot appears, indicating the breakpoint.



```
import java.util.StringTokenizer;
import javax.swing.JOptionPane;

/**
  This program tests the countSyllables method of the Word class.
*/
public class WordTest
{
  public static void main(String[] args)
  {
    String input
                                                     tence");
    StringTokenizer tokenizer = new StringTokenizer(input);
    while (tokenizer.hasMoreTokens())
    {
      String token = tokenizer.nextToken();
      Word w = new Word(token);
      int syllables = w.countSyllables();
      System.out.println("Syllables in " + token + ": "
        + syllables);
    }
    System.exit(0);
  }
}
```

Then select the menu option **Run->Debug as...->Java Application**. The debugger starts and switches the Eclipse display to the *debugger perspective*. It pauses at the breakpoint that you set.



Debug - WordTest.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
WordTest.java
Outline
Console Tasks

```
<terminated> /usr/local/j2sdk1.4.1_01/bin/java (3/25/03 9:34 AM)
WordTest [Java Application]
  WordTest at localhost:5508
    System Thread [Finalizer] (Running)
    System Thread [Reference Handler] (Running)
    Thread [main] (Suspended (breakpoint at line 12 in WordTest))
      WordTest.main(String[]) line: 12
```

```
args= String[0]  (id=14)
```

```
public static void main(String[] args)
{
  String input
    = JOptionPane.showInputDialog("Enter a sentence");
  StringTokenizer tokenizer = new StringTokenizer(input);
  while (tokenizer.hasMoreTokens())
  {
    String token = tokenizer.nextToken();
    Word w = new Word(token);
    int syllables = w.countSyllables();
```

```
import declarations
WordTest
```

Console [WordTest at localhost:5508]

Writable | Insert | 12 : 60

Whenever you select the menu option **Run->Step Over** (or the `F6` keyboard shortcut), then the debugger executes one line of the program, *without stepping inside method calls*. For example, tracing over the call

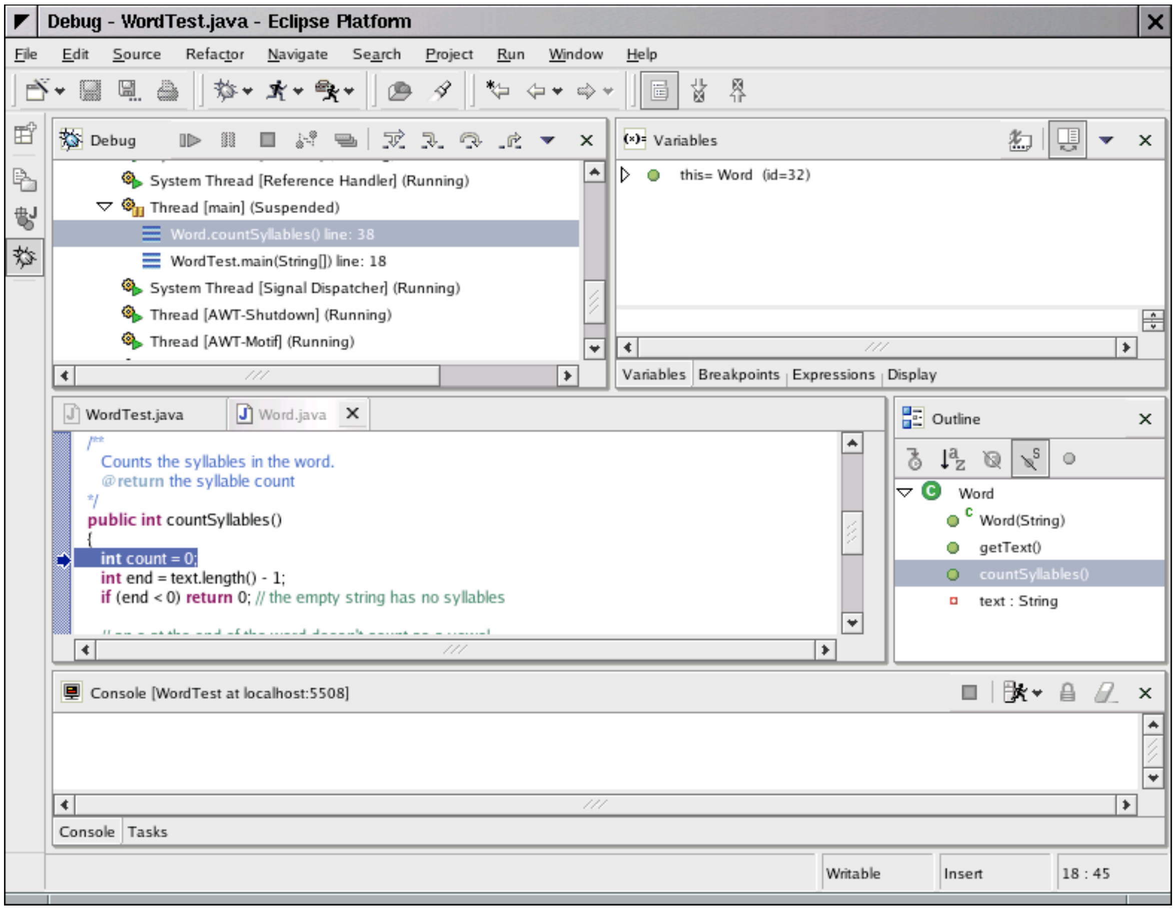
```
Word w = new Word(token);
```

will not trace inside the `Word` constructor but simply run the program to the next line of the `main` method.

Contrast that with the menu option **Run->Step Into**(or the `F5` keyboard shortcut). This command traces inside method calls. For example, tracing into the line

```
int syllables = w.countSyllables();
```

stops at the first line of the `countSyllables` method:


Debug - WordTest.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
WordTest.java
Word.java
Outline
Console Tasks
Writable
Insert
18 : 45

```
System Thread [Reference Handler] (Running)
Thread [main] (Suspended)
    Word.countSyllables() line: 38
    WordTest.main(String[]) line: 18
System Thread [Signal Dispatcher] (Running)
Thread [AWT-Shutdown] (Running)
Thread [AWT-Motif] (Running)
```

```
this= Word (id=32)
```

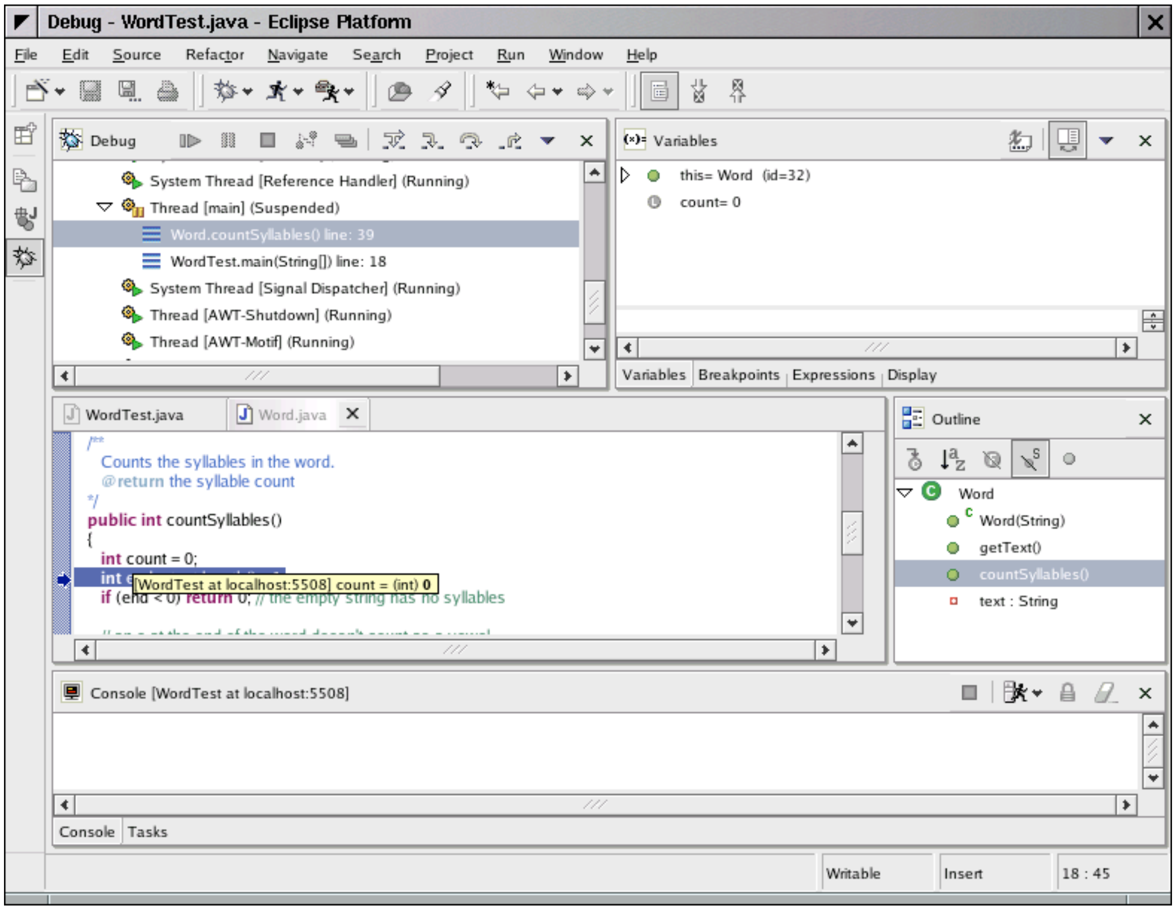
```java
/**
   Counts the syllables in the word.
   @return the syllable count
*/
public int countSyllables()
{
   int count = 0;
   int end = text.length() - 1;
   if (end < 0) return 0; // the empty string has no syllables
```

```
Word
    Word(String)
    getText()
    countSyllables()
    text : String
```

Console [WordTest at localhost:5508]

Watching values

To see the value of a variable of a simple type (such as a number or a string), simply hold the mouse pointer over the variable name when the debugger is stopped. Then the contents of the variable is displayed in a small rectangle next to the variable name. For example, here is the contents of the `count` variable in the `countSyllables` method.


Debug - WordTest.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
WordTest.java
Word.java
Outline
Console [WordTest at localhost:5508]
Console Tasks
Writable
Insert
18 : 45

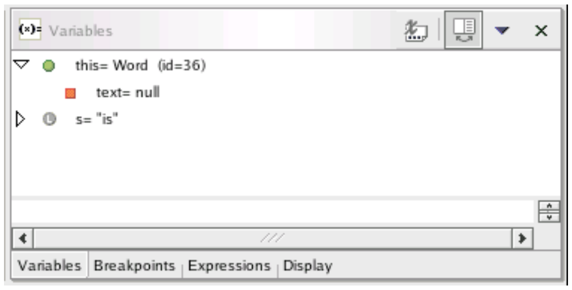
```
System Thread [Reference Handler] (Running)
Thread [main] (Suspended)
    Word.countSyllables() line: 39
    WordTest.main(String[]) line: 18
System Thread [Signal Dispatcher] (Running)
Thread [AWT-Shutdown] (Running)
Thread [AWT-Motif] (Running)
```

```
this= Word  (id=32)
count= 0
```

```
/**
   Counts the syllables in the word.
   @return the syllable count
*/
public int countSyllables()
{
   int count = 0;
   int e[WordTest at localhost:5508] count = (int) 0
   if (end < 0) return 0; // the empty string has no syllables
```

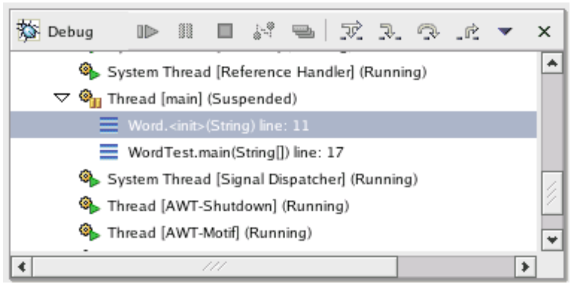
```
Word
   Word(String)
   getText()
   countSyllables()
   text : String
```

The top right window shows all local variables. Click on the triangles to look inside objects. To see the instance fields of the implicit parameter, look inside `this`.



```
this= Word  (id=36)
   text= null
s= "is"
```

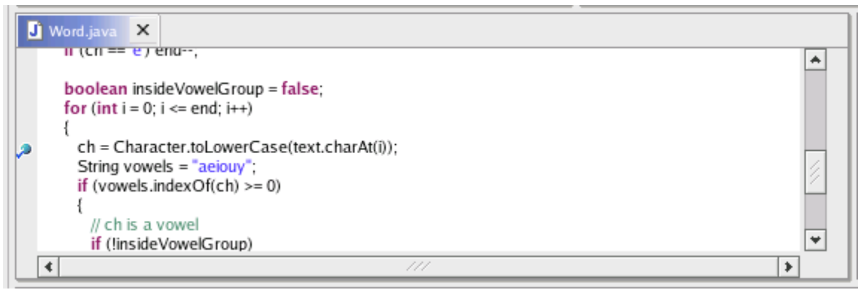
The top left window shows the call stack (with the most recently called method on top). For example, the following call stack shows that the `WordTest.main` method called the `Word.<init>` method, that is, the constructor.



```
System Thread [Reference Handler] (Running)
Thread [main] (Suspended)
    Word.<init>(String) line: 11
    WordTest.main(String[]) line: 17
System Thread [Signal Dispatcher] (Running)
Thread [AWT-Shutdown] (Running)
Thread [AWT-Motif] (Running)
```

Setting breakpoints

Tracing through a program can be tedious. Often, you want the program to run at full speed until a particular line of code is encountered. To set a *breakpoint* at a line, double-click on the gray bar to the left. A blue dot indicates the breakpoint.



```
    boolean insideVowelGroup = false;
    for (int i = 0; i <= end; i++)
    {
      ch = Character.toLowerCase(text.charAt(i));
      String vowels = "aeiouy";
      if (vowels.indexOf(ch) >= 0)
      {
        // ch is a vowel
        if (!insideVowelGroup)
```

Now select the menu option **Run->Resume** or hit the F8 keyboard shortcut. The program runs to the next breakpoint, stopping only for user input.

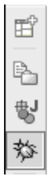
You can set as many breakpoints as you like.

To remove a breakpoint, double-click on it with the mouse.

Stopping the debugger

When the program has completed, the debugger stops automatically. When you want to terminate a debugging session without running the program to the end, select the menu option **Run->Terminate**.

To return to the Java perspective (in which you can edit and compile your program), locate the perspectives toolbar at the left of the Eclipse frame:



Click on the button with the "J" (above the bug button). Eclipse will remove the debugger windows and restore the edit windows.