# Installing Eclipse [1, 2]

**by Christopher Batty and David Scuse**

**Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada**

**Last revised: October 22, 2003**

## Overview:

In this collection of documents, we describe how to develop Java applications that use the SWT (Standard Widget Toolkit) instead of using Java's Swing or AWT widgets. SWT applications are developed using the Eclipse workbench. By the end of this document, you will be able to create a stand-alone Java application (jar file) that uses the SWT and runs on either Windows or Linux.

In this document, we describe the installation of Eclipse on Windows and Linux platforms and the creation of a simple Java program that uses the SWT. We have attempted to note common problems that affect the install and provide solutions to these problems. We used Windows XP Professional with the Java 1.4.0 JRE and Eclipse 2.1 as the primary platform for illustrating the use of Eclipse, but issues specific to Linux (both GTK and Motif) are also described at the end of this document.

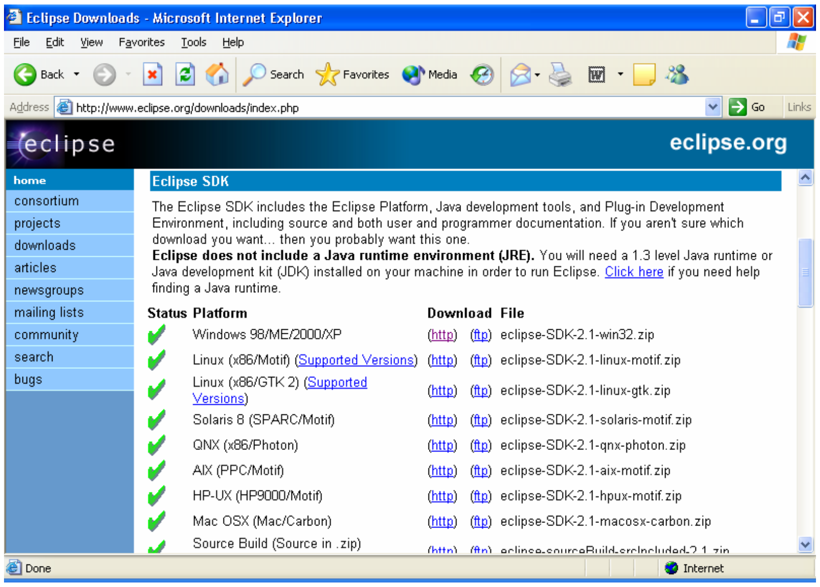
(c) Copyright IBM Corp. and others. 2000, 2003. All rights reserved. Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S., other countries, or both.

## Setting up Eclipse

To begin, download the "Latest Release" version of Eclipse from the downloads page of the Eclipse website located at http://eclipse.org. (If you would like to try using the most recent stable version of Eclipse containing the latest new features, you can install the

[1] This work was funded by an IBM Eclipse Innovation Grant.

"Current Stable Build".) Ensure that you select the correct version for your particular setup. The download file should have a name similar to "eclipse-SDK-2.1-win32.zip" or "eclipse-SDK-2.1-linux-motif.zip" depending on the operating system and windowing system you use. For each release of Eclipse, a page identifies the downloads that are available. Note that the source code for the release is available at the end of the list.

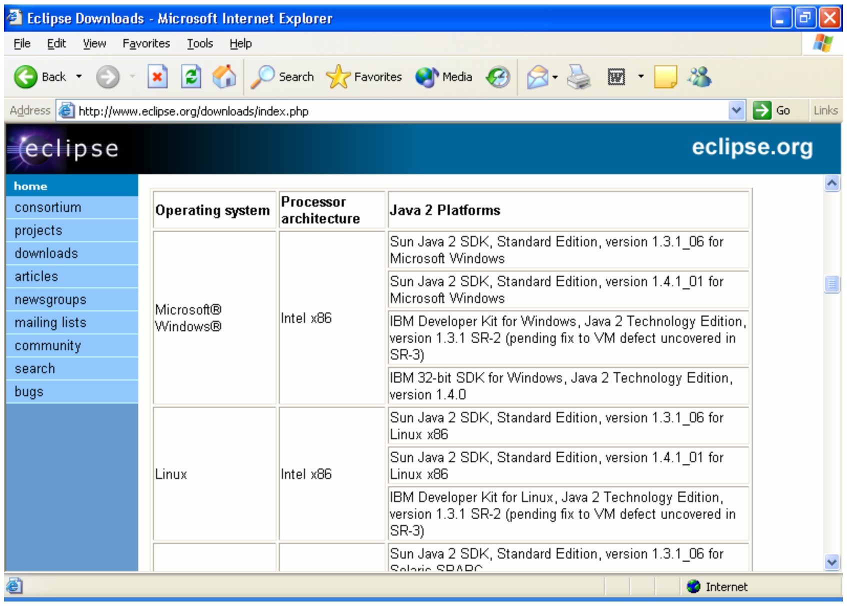

Eclipse Downloads - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.eclipse.org/downloads/index.php Go Links
eclipse
eclipse.org
home
consortium
projects
downloads
articles
newsgroups
mailing lists
community
search
bugs
Done
Internet

**Eclipse SDK**

The Eclipse SDK includes the Eclipse Platform, Java development tools, and Plug-in Development Environment, including source and programmer documentation. If you aren't sure which download you want... then you probably want this one.
**Eclipse does not include a Java runtime environment (JRE).** You will need a 1.3 level Java runtime or Java development kit (JDK) installed on your machine in order to run Eclipse. Click here if you need help finding a Java runtime.

| Status | Platform | Download | File |
|---|---|---|---|
| ✓ | Windows 98/ME/2000/XP | (http) (ftp) | eclipse-SDK-2.1-win32.zip |
| ✓ | Linux (x86/Motif) (Supported Versions) | (http) (ftp) | eclipse-SDK-2.1-linux-motif.zip |
| ✓ | Linux (x86/GTK 2) (Supported Versions) | (http) (ftp) | eclipse-SDK-2.1-linux-gtk.zip |
| ✓ | Solaris 8 (SPARC/Motif) | (http) (ftp) | eclipse-SDK-2.1-solaris-motif.zip |
| ✓ | QNX (x86/Photon) | (http) (ftp) | eclipse-SDK-2.1-qnx-photon.zip |
| ✓ | AIX (PPC/Motif) | (http) (ftp) | eclipse-SDK-2.1-aix-motif.zip |
| ✓ | HP-UX (HP9000/Motif) | (http) (ftp) | eclipse-SDK-2.1-hpux-motif.zip |
| ✓ | Mac OSX (Mac/Carbon) | (http) (ftp) | eclipse-SDK-2.1-macosx-carbon.zip |
| ✓ | Source Build (Source in .zip) | (http) (ftp) | eclipse-sourceBuild-srcIncluded-2.1.zip |

The link to the Supported Versions page provides more information on the specific requirements necessary for Eclipse.


Eclipse Downloads - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.eclipse.org/downloads/index.php Go Links
eclipse
eclipse.org
home
consortium
projects
downloads
articles
newsgroups
mailing lists
community
search
bugs
Internet

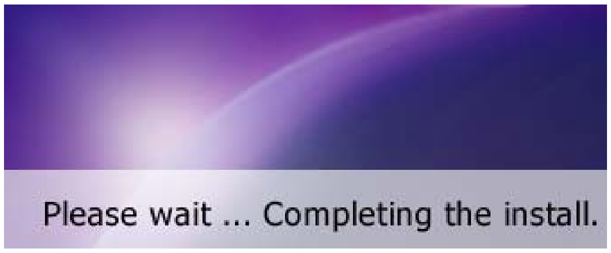
| Operating system | Processor architecture | Java 2 Platforms |
|---|---|---|
| Microsoft® Windows® | Intel x86 | Sun Java 2 SDK, Standard Edition, version 1.3.1_06 for Microsoft Windows |
| | | Sun Java 2 SDK, Standard Edition, version 1.4.1_01 for Microsoft Windows |
| | | IBM Developer Kit for Windows, Java 2 Technology Edition, version 1.3.1 SR-2 (pending fix to VM defect uncovered in SR-3) |
| | | IBM 32-bit SDK for Windows, Java 2 Technology Edition, version 1.4.0 |
| Linux | Intel x86 | Sun Java 2 SDK, Standard Edition, version 1.3.1_06 for Linux x86 |
| | | Sun Java 2 SDK, Standard Edition, version 1.4.1_01 for Linux x86 |
| | | IBM Developer Kit for Linux, Java 2 Technology Edition, version 1.3.1 SR-2 (pending fix to VM defect uncovered in SR-3) |
| | | Sun Java 2 SDK, Standard Edition, version 1.3.1_06 for |

The download file is in .zip format and can be expanded with any standard decompression software. Expand it into the folder where you would like Eclipse to run from, (e.g. "C:\Program Files\eclipse"). Note: **There is no formal installation**

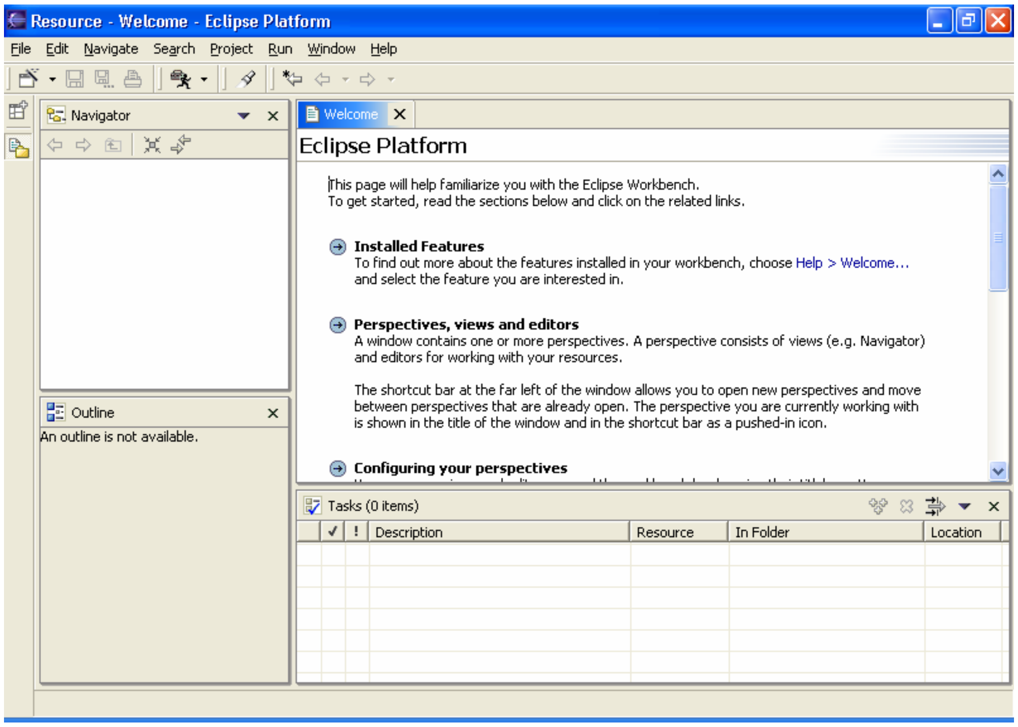
**executable to run.** To remove Eclipse you simply delete the Eclipse folder, because Eclipse does not alter the system registry.

Eclipse requires that a Java Runtime Environment (JRE) be installed on your system. The minimum version is identified on the Supported Versions page. (A JDK comes with a JRE, so if you can write and run Java programs, you already have a suitable JRE.) Entering the command **`java -version`** at the command prompt displays the version of the JRE. You can download and install Sun's Java 2 Standard Edition SDK for your particular operating system, available from Sun's website at http://java.sun.com.

Once you have a Java runtime environment installed, you can simply run the Eclipse executable located in the install directory. (If you do not have a Java runtime, Eclipse will display an error message and refuse to run.) When Eclipse is run for the first time, the following screen is displayed for a short period of time.


Please wait ... Completing the install.

Then, after the Eclipse splash screen (shown on page 1) is displayed, the Eclipse workbench screen is displayed.


Resource - Welcome - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline
Welcome
Tasks (0 items)
Description
Resource
In Folder
Location

An outline is not available.

Eclipse Platform

This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related links.

**Installed Features**
To find out more about the features installed in your workbench, choose Help > Welcome...
and select the feature you are interested in.

**Perspectives, views and editors**
A window contains one or more perspectives. A perspective consists of views (e.g. Navigator)
and editors for working with your resources.

The shortcut bar at the far left of the window allows you to open new perspectives and move
between perspectives that are already open. The perspective you are currently working with
is shown in the title of the window and in the shortcut bar as a pushed-in icon.

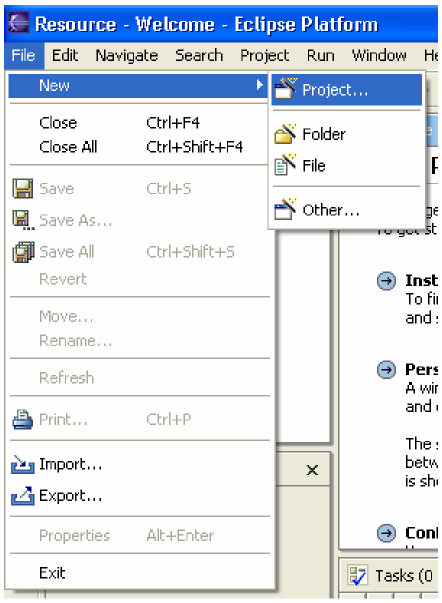
**Configuring your perspectives**

The “Navigator”, “Outline”, and “Tasks” areas of the screen are known as “views”, while the area labeled “Welcome” is called an editor. In general, views provide information and allow you to navigate and modify properties of files, classes, packages, etc. Editors are designed to allow you to edit the contents of various types of files. For example, you will write your Java code in a Java editor window.

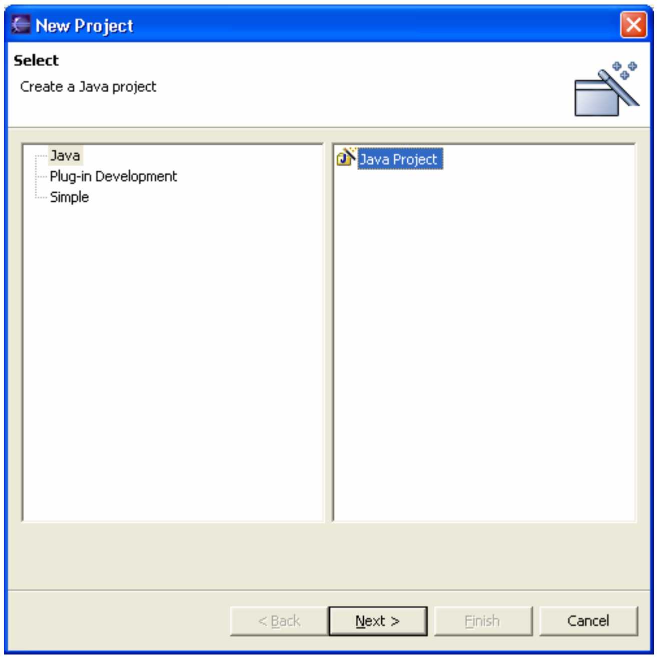
The combination of views and editors displayed on screen at a given time is called a “perspective”. The current perspective is called the “Resource Perspective” and is used to display and edit resources. The currently open perspectives are shown in the toolbar down the left side of the screen. Other perspectives that will be useful later include the “Java Perspective” for editing java files, and the “Debug Perspective,” for performing debugging tasks.

## Creating a Project

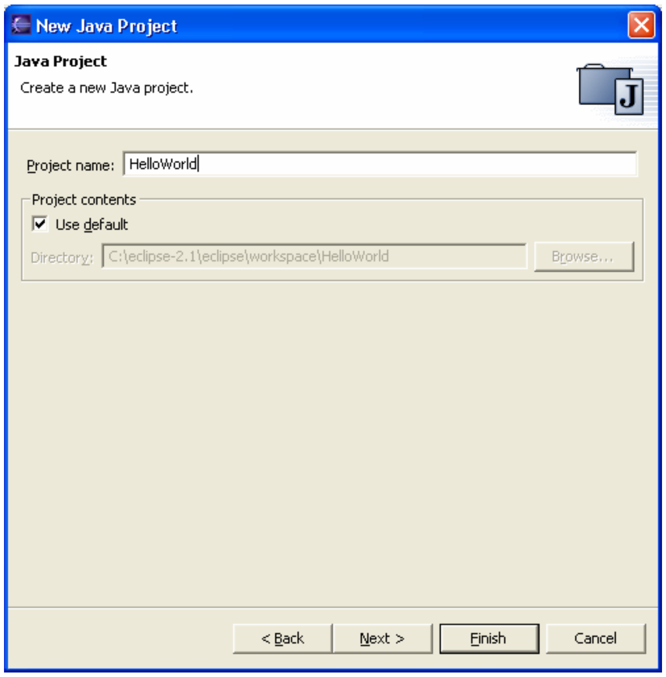
To begin developing a Java program using Eclipse, you must first create a new project. From the File menu, select New -> Project.



A New Project Wizard is displayed. On the first page, select Java in the tree list on the left, and choose Java Project from the choices on the right. Then select the Next button.



In the “Project name” field, enter a name for your new project. If you enter invalid characters, an error message is displayed at the top area of the screen. All Eclipse Wizards display errors in this manner. You may choose to store your project in the default folder location which will be under <eclipse-install-directory>\workspace\<project name> as in the following figure. Alternatively, you can uncheck the “Use default” checkbox, and specify another location for your project. When you have filled in these fields, hit the Next button.



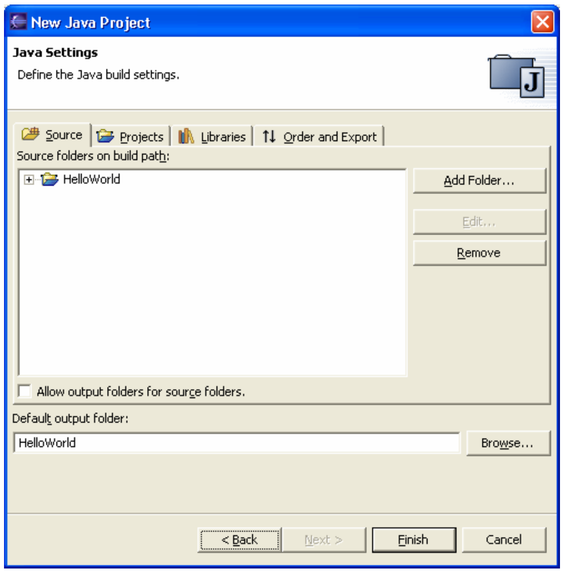
Project name: HelloWorld

Project contents

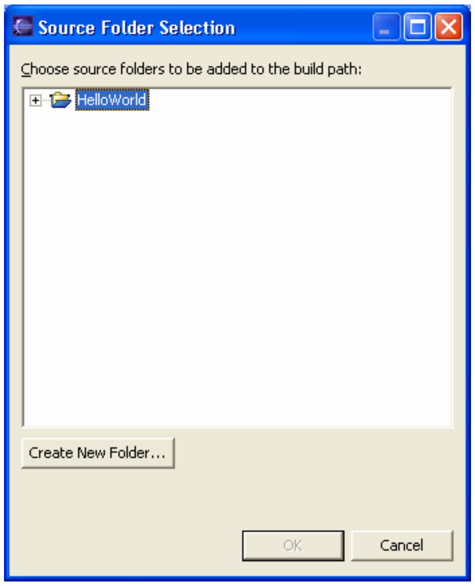
☑ Use default

Directory: C:\eclipse-2.1\eclipse\workspace\HelloWorld

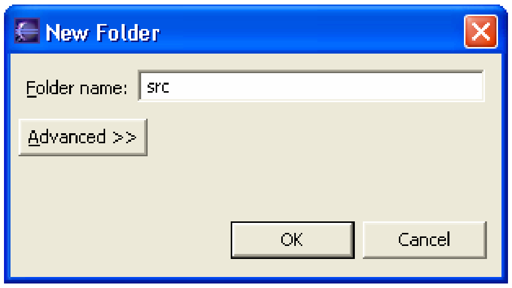
The next page of the wizard allows you to modify various properties of the project. For the time being, the only tab we will worry about is the "Source" tab. This determines how the various files will be stored within your project folder. The "Projects", "Libraries" and "Order and Export" tabs deal with how Java locates and uses other projects and libraries that may be required for the project you are creating. For example, if we make use of the SWT user interface library, we must specify this under the Libraries tab (configuring SWT will be covered later in this document). These settings can all be modified later from the project's Properties dialog.



The Eclipse source files are normally stored in their own directory in the project directory. Click the Add Folder button to create a separate source folder.

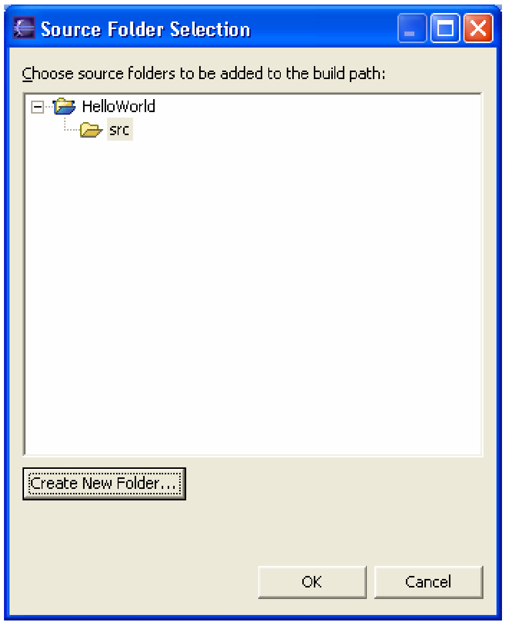


Enter the name of the folder in which to store the .java files. A standard choice for this is usually "src".



Folder name: src

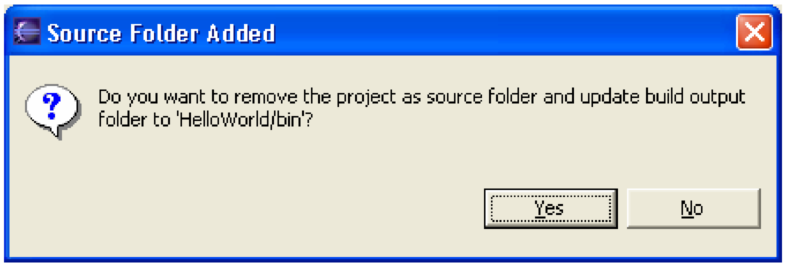
The source folder is now displayed in the project hierarchy.



Choose source folders to be added to the build path:

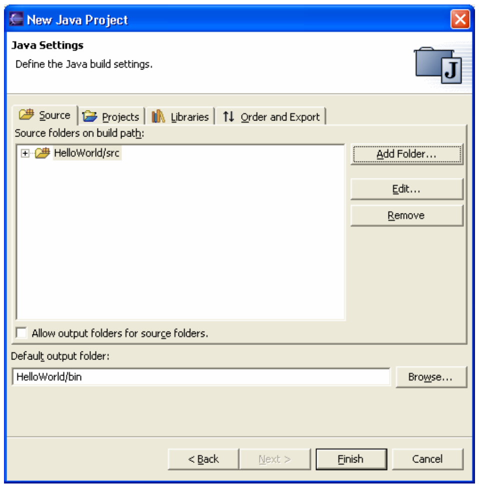
HelloWorld
src

Eclipse will then ask if you would like to update the output folder to "<project name>/bin". This is where your output (usually *.class) files will be placed upon compilation. Choose "Yes".

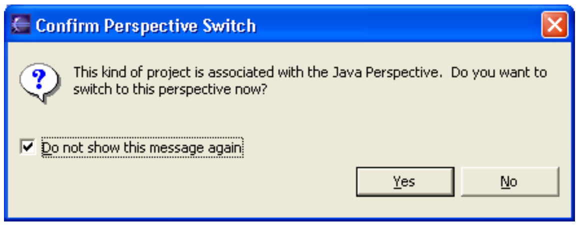


Do you want to remove the project as source folder and update build output folder to 'HelloWorld/bin'?

The screen should now look like the following.



To finish creating the project, you may now hit "Finish". Eclipse now asks if you want to switch to the Java Perspective in the workbench.

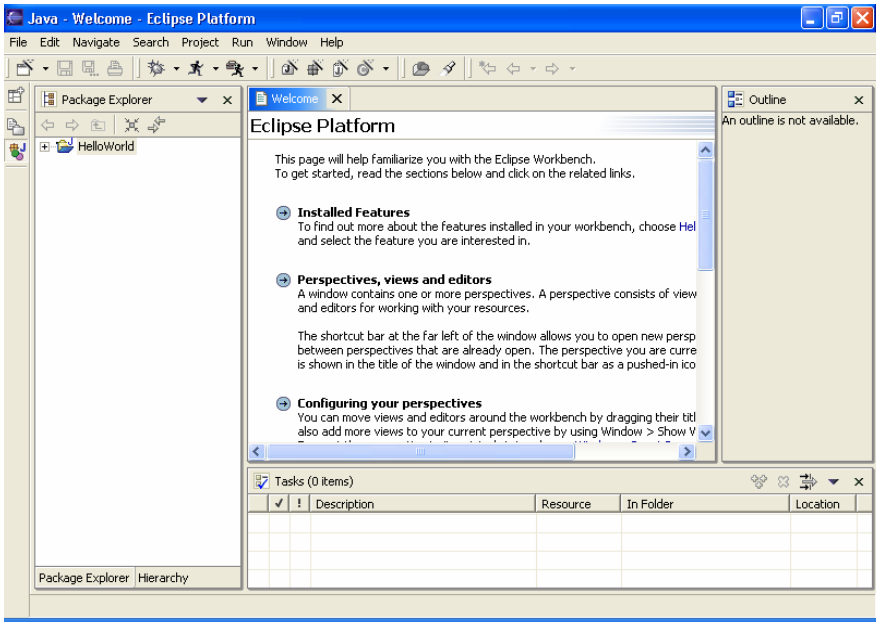

Confirm Perspective Switch

This kind of project is associated with the Java Perspective. Do you want to switch to this perspective now?

[✓] Do not show this message again

## Creating a Java program

When you finish creating the new Java project, the screen should look like the following.



This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related links.

**Installed Features**
To find out more about the features installed in your workbench, choose Hel
and select the feature you are interested in.

**Perspectives, views and editors**
A window contains one or more perspectives. A perspective consists of view
and editors for working with your resources.

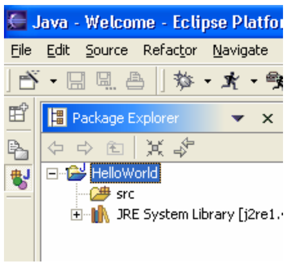
The shortcut bar at the far left of the window allows you to open new persp
between perspectives that are already open. The perspective you are curre
is shown in the title of the window and in the shortcut bar as a pushed-in ico

**Configuring your perspectives**
You can move views and editors around the workbench by dragging their titl
also add more views to your current perspective by using Window > Show V

There are several things to note about this. The view on the left of the screen is now the "Package Explorer", and the "Outline" view is on the right rather than the left. The reason for this is that we are now looking at a different "perspective". Recall that a perspective is a combination of view and editor windows displayed on the screen. The currently displayed perspective is the "Java Perspective", usually used for editing java files. The perspective seen earlier was the "Resource Perspective".

Notice also that a new button has been added to the tool bar down the left side of the screen. This button, which has a few shapes and a "J" on it, represents the "Java Perspective." To switch between perspectives, simply click the button for the perspective you'd like to see.

More important than the change in perspective is the fact that our new project is now listed in the Package Explorer view on the left. We will use this view to navigate the packages, source files and libraries that comprise our project. If we click the "+" sign on the tree beside the project name, and expand it we should see two subfolders displayed.


Java - Welcome - Eclipse Platfo
File Edit Source Refactor Navigate
Package Explorer
HelloWorld
src
JRE System Library [j2re1.

The first is labeled "src." This is the folder in which our source java files will be placed.

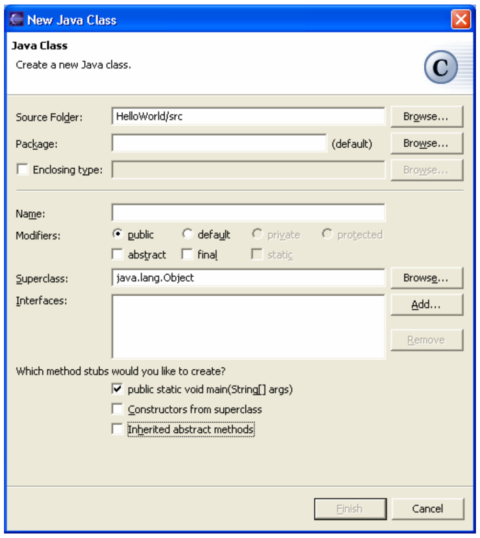
The second may be labeled differently, depending on the JRE that is installed. In the example to the left, the full path is:

```
JRE_LIB – C:\Program Files\Java\j2re1.4.1_03\lib\rt.jar
```

This is the location of the JAR (Java archive) file that contains Sun's Java Libraries on this machine. If you expand it further, you can browse the packages and classes of the libraries. We can essentially ignore this library for now.

The next thing we do is create a new class for our first program. Select the "src" folder in the Package Explorer. From the File menu, choose New -> Class. Alternatively, the button on the toolbar that looks like will also run the "New Java Class" wizard. (Note that you must be in the Java Perspective for this portion of the toolbar to be displayed.) Many other useful functions have buttons on the toolbar as well. For example, pressing is a quick way to run the "New Java Project" wizard.

When the “New Java Class” wizard runs you should see the following screen:



**Java Class**

Create a new Java class.

Source Folder: HelloWorld/src

Package: (default)

☐ Enclosing type:

Name:

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

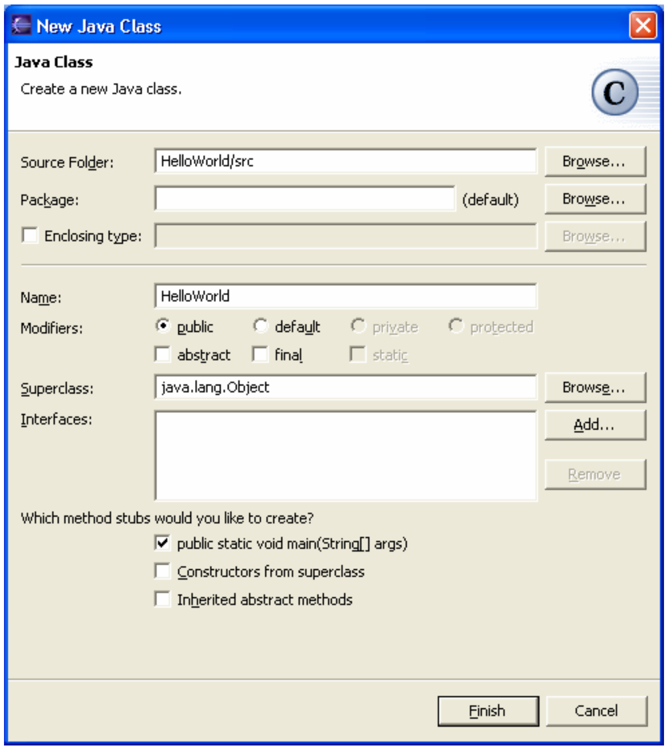
Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☐ Inherited abstract methods

Enter the name of your class in the “Name” field.



**Java Class**

Create a new Java class.

Source Folder: HelloWorld/src

Package: (default)

☐ Enclosing type:

Name: HelloWorld

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

The “Source Folder” should be <project name>/src already, so leave it as is. (If the correct folder was not selected when you ran the Wizard, it may be incorrect and if so you should change it to reflect the correct location for your source files.)

In the “Package” field, if you wish to put the new class in a particular package, you can browse for the package to use, or specify its name. If the package you specify doesn’t exist, a new package by that name will be created. For our purposes we can simply use the default package, leaving the field blank.

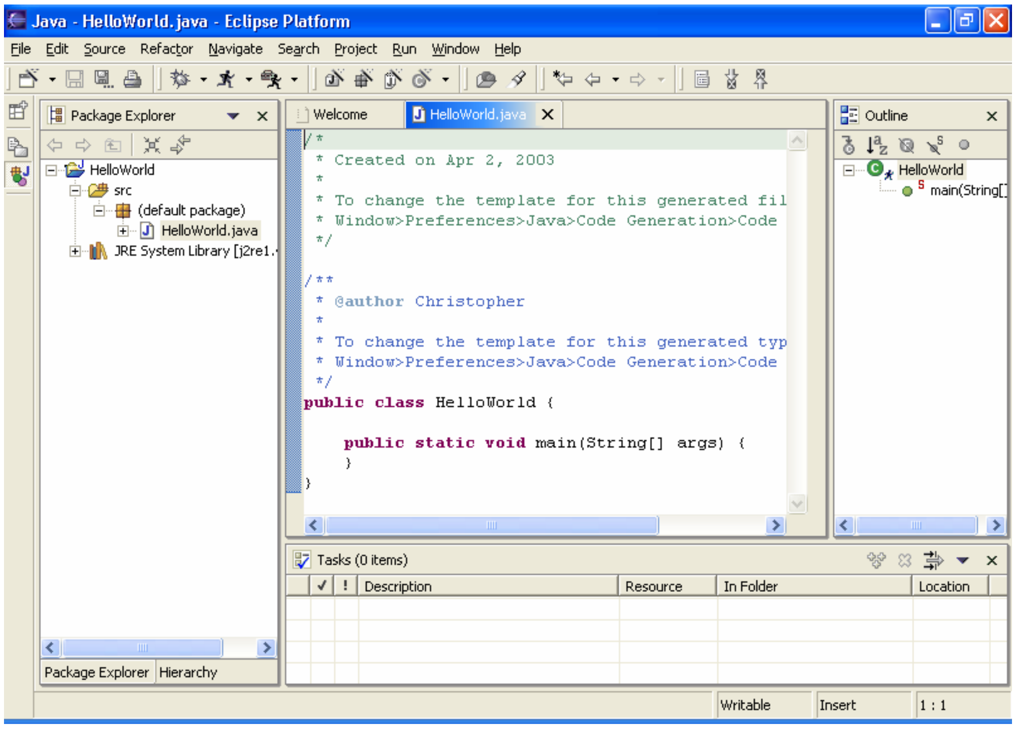
The “Enclosing type” field is only used if you are creating an “inner class,” which we are not.

Using this wizard we can also select access specifiers and identify superclasses or interfaces we wish to extend or implement. Use the default settings.

One feature of the wizard that can be useful is the auto-generation of method stubs (empty methods with the correct names, parameters and return values). If we were implementing or extending an interface or class, stubbing the inherited classes would be very helpful, since we would not have to continually refer back to the superclass to determine the method signatures. For now, check only “public static void main (String[] args)” which will stub the main method for us.

If there were any errors in how you have set up your class, a descriptive error message will be displayed at the top of the page. Hit “Finish” to generate the new class.

The screen should now look like this:



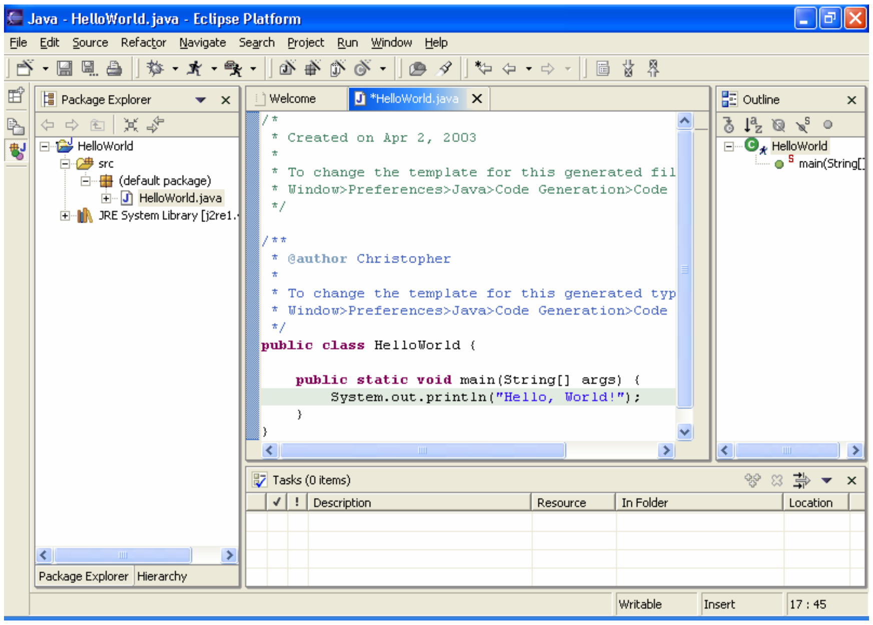
```
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated fil
 * Window>Preferences>Java>Code Generation>Code
 */

/**
 * @author Christopher
 *
 * To change the template for this generated typ
 * Window>Preferences>Java>Code Generation>Code
 */
public class HelloWorld {

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

You can see that in the central (editor) window, there is a new tab, with the title “HelloWorld.java” corresponding to the name of the class specified in the wizard. To switch between open files/documents, simply click the various tabs. To close one of the files, click the tab so that the file is selected, and hit the ‘X’ button on the tab. To re-open a closed file, you can double-click on its name in the Package Explorer or Navigator views.

Notice that in the Outline view, an outline of the currently selected class is displayed. It will show all fields and methods of the class. Double-clicking on a method/field will jump to its definition in the source code. The buttons at the top of the view allow you to set options such as displaying only public members, sorting the names alphabetically, or hiding fields.

Edit the main() function of the HelloWorld class to contain the following line:

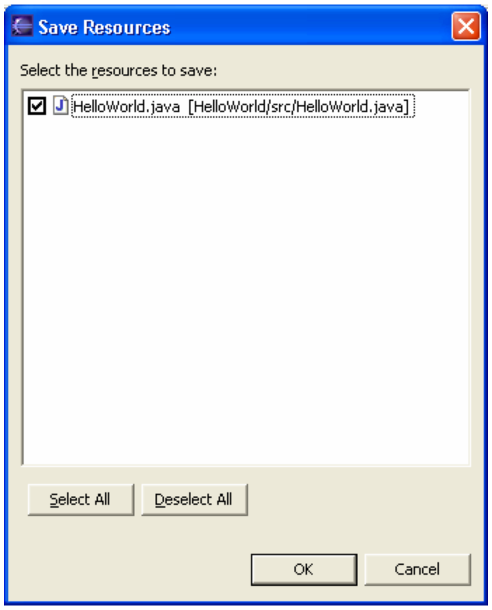
**`System.out.println(“Hello, World!”);`**



```
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated fil
 * Window>Preferences>Java>Code Generation>Code
 */

/**
 * @author Christopher
 *
 * To change the template for this generated typ
 * Window>Preferences>Java>Code Generation>Code
 */
public class HelloWorld {

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

We will use this simple program to verify that we can successfully run a basic Java program using the Eclipse environment.

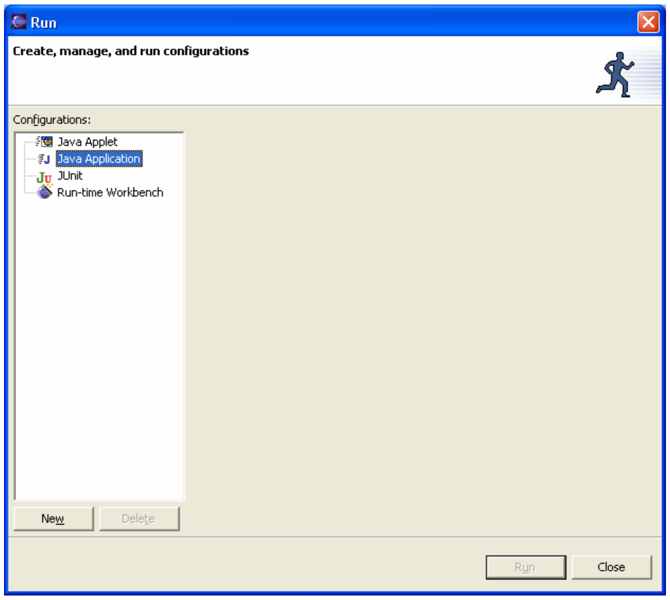
From the File menu, choose “Save” to save the file. When you save the file, Eclipse automatically compiles it, and updates the .class files. Note that if you do forget to save a file before running a program, Eclipse will prompt you to ensure that you have saved all the necessary files. The dialog for that looks like the following:


Save Resources
Select the resources to save:
HelloWorld.java [HelloWorld/src/HelloWorld.java]
Select All
Deselect All
OK
Cancel

Check the boxes for each file/resource you wish to save before running the program. Again, compilation will be performed, so the latest changes will be reflected in the .class files for the program. (For small projects, this automatic compilation is generally quite useful, but it can become a nuisance for **very** large projects, so there is a preference available to turn it off under Window->Preferences. Choose the Workbench in the tree on the left and uncheck "Perform build automatically on resource modification." Ctrl+B or Projects->Rebuild All can be used to rebuild the project when necessary.)

## Running a Java program

From the Run menu, select the "Run…" option. You will see the following screen:



This screen allows you to set up “Launch Configurations”. The general idea is to set up all the parameters and options for a particular run of your program. If there are different “set-ups” that you use repeatedly, such as running with different command-line parameters or different java libraries or runtimes, you can create a launch configuration for each set-up. This way, you can select the correct launch configuration, and the program will run with the correct parameters, rather than having to modify the parameters each time you run the program.

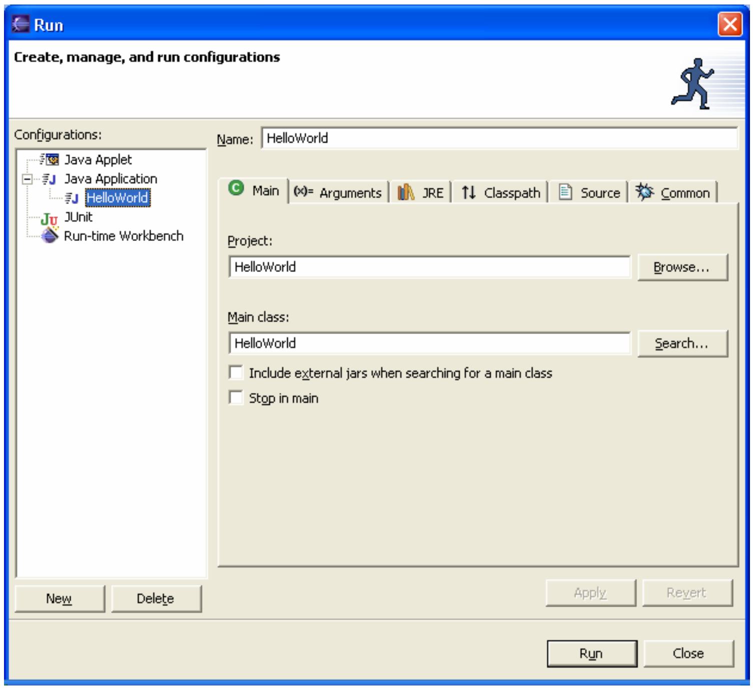
Select the “Java Application” option as above and hit the “New” button. A new Launch Configuration for a Java Application will be created and displayed (both on the right and in the tree-view on the left).

For those who are interested in the other options:

- JUnit is a popular testing framework that is used to create automated tests for your Java classes. See [http://junit.org](http://junit.org) for more information.

- Run-time Workbench is used by developers who are creating plug-ins for Eclipse. It runs a new instance of the Eclipse “Work-bench” for testing the new plug-in.

Assuming that the correct class was selected in the Package Explorer, most of the settings should be filled in automatically, as seen below. If the class or project names are incorrect, you can hit Search or Browse, respectively, to choose the correct name(s) from a list. By default, the name of the launch configuration be the currently active class.



Create, manage, and run configurations

Name: HelloWorld

Project: HelloWorld

Main class: HelloWorld

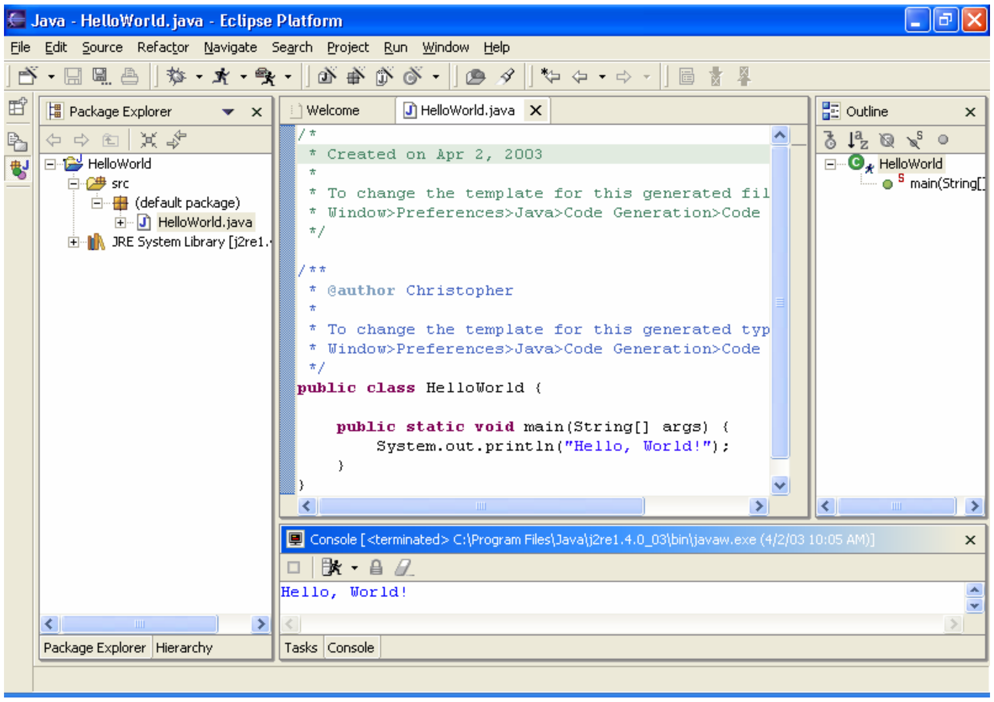
☐ Include external jars when searching for a main class

☐ Stop in main

The **Arguments** tab allows you to specify command-line arguments to either the Java virtual machine (that your program runs on) or to your program. The **JRE** tab allows you to change which Java Runtime Environment you are using to run the program. The **Classpath** tab is for setting the paths on your system where the program will look for class files to run the program. The **Source** tab tells the system where to look for program source files, and finally the **Common** tab sets various options dealing with shared (versus local) launch configurations.

The defaults for these other tabs will work fine for now. To continue, hit Run.

The screen should look as follows after running the program. The console output is displayed in a new "Console" view at the bottom of the screen. Note that the name of JRE used to run the application is displayed at the top of the console window.

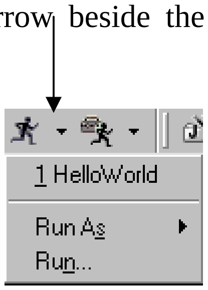


Java - HelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
HelloWorld
src
(default package)
HelloWorld.java
JRE System Library [j2re1.
Welcome
HelloWorld.java
Outline
HelloWorld
main(String[]
Console [<terminated> C:\Program Files\Java\j2re1.4.0_03\bin\javaw.exe (4/2/03 10:05 AM)]
Package Explorer
Hierarchy
Tasks
Console

```
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated fil
 * Window>Preferences>Java>Code Generation>Code
 */

/**
 * @author Christopher
 *
 * To change the template for this generated typ
 * Window>Preferences>Java>Code Generation>Code
 */
public class HelloWorld {

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

```
Hello, World!
```

Once a Launch Configuration has been defined, it is not necessary to go through the Run command each time that an application is run. Instead, click the run icon to run the current launch configuration (or use Ctrl-F11). If multiple launch configurations have been defined, clicking the down arrow beside the run icon displays a list of launch configurations that can be selected.



## Configuring Eclipse to use the SWT libraries

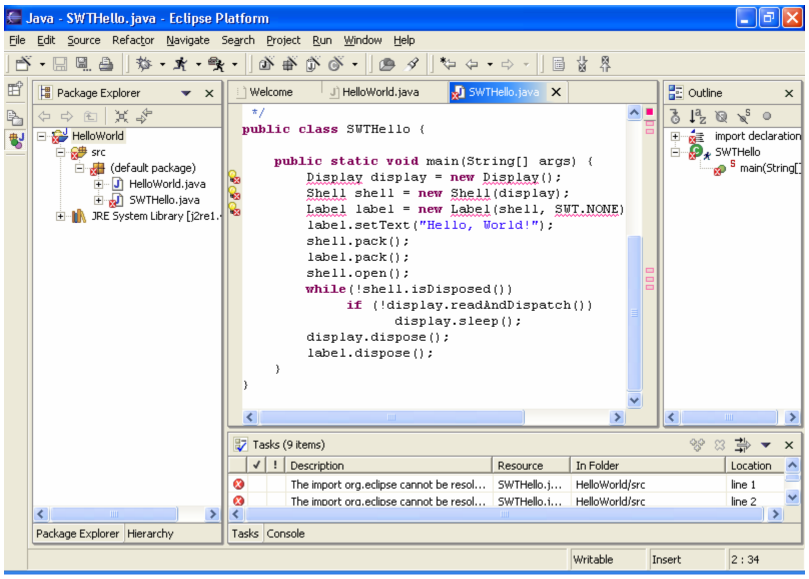
In the previous section, we configured the Eclipse workbench to run a simple Java program. In fact, any Java program that uses only Sun’s Java run-time libraries (including Swing and the AWT) can be run with this configuration. However, we are interested in using the Eclipse SWT (Standard Widget Toolkit) instead of Java’s Swing and AWT GUI widgets. In this section, we describe how to set up Eclipse to use the SWT user interface libraries and to verify that the configuration works correctly.

To begin, create a new class called SWTHello in the existing HelloWorld Project. Replace the auto-generated code with the following:

```java
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;
public class SWTHello {
      public static void main(String[] args) {
            Display display = new Display();
            Shell shell = new Shell(display);
            Label label = new Label(shell, SWT.NONE);
            label.setText("Hello, World!");
            shell.pack();
            label.pack();
            shell.open();
            while(!shell.isDisposed())
                  if(!display.readAndDispatch())
                        display.sleep();
            display.dispose();
            label.dispose();
      }
}
```

Use the File menu (or Ctrl-S) to save the file. This should cause numerous errors to be displayed. Notice that the errors are displayed in several locations, typically using small red circles containing white ‘X’s.

1. The Package Explorer identifies each branch of the project tree leading to an offending class file(s).
2. The Outline view shows the methods in which the errors occur.
3. The central Java editor window signifies errors by “squiggly” red underlines, similar to the mechanism used in Microsoft Word to identify spelling errors.
4. Lastly, the Task view at the bottom of the screen now includes a list of tasks that are the errors returned by the Java compiler. Double-clicking the error message will move your cursor to the location of the related error in the editor window. You may also find the Task view useful for organizing your work. To add a new Task to the list, right-click in the list (or use the button on the right side Task view title bar).



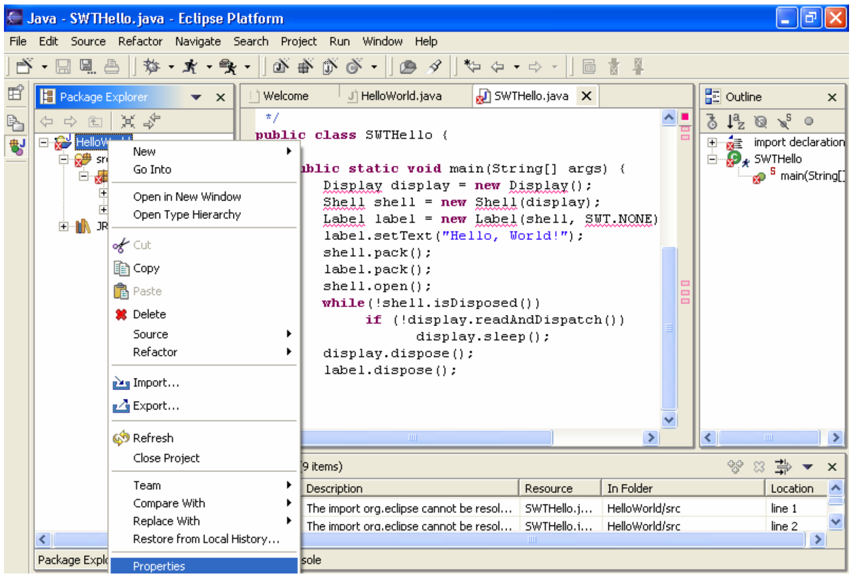
```
*/
public class SWTHello {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        Label label = new Label(shell, SWT.NONE);
        label.setText("Hello, World!");
        shell.pack();
        label.pack();
        shell.open();
        while(!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
        label.dispose();
    }
}
```

Tasks (9 items)

| | ✓ | ! | Description | Resource | In Folder | Location |
|---|---|---|---|---|---|---|
| ⊗ | | | The import org.eclipse cannot be resol... | SWTHello.j... | HelloWorld/src | line 1 |
| ⊗ | | | The import org.eclipse cannot be resol... | SWTHello.j... | HelloWorld/src | line 2 |

The source of these errors is that we have not included the SWT libraries in the project, so the compiler is unsuccessfully searching for the SWT packages and classes that the program requires. To resolve this, we must supply the program with the locations of both the Java classes for the SWT libraries and the “native” (OS-dependent) code that the SWT classes use at run-time. The classes are in swt.jar and the run-time routines are in the SWT DLL file. (The name of this DLL depends on the platform and current SWT version. In our configuration it is “swt-win32-2133.dll”.)

To indicate where to find swt.jar, first right-click on the project name in the Package Explorer, and choose Properties. This will bring up a Properties dialog containing many of the same settings that were available when creating the Project.

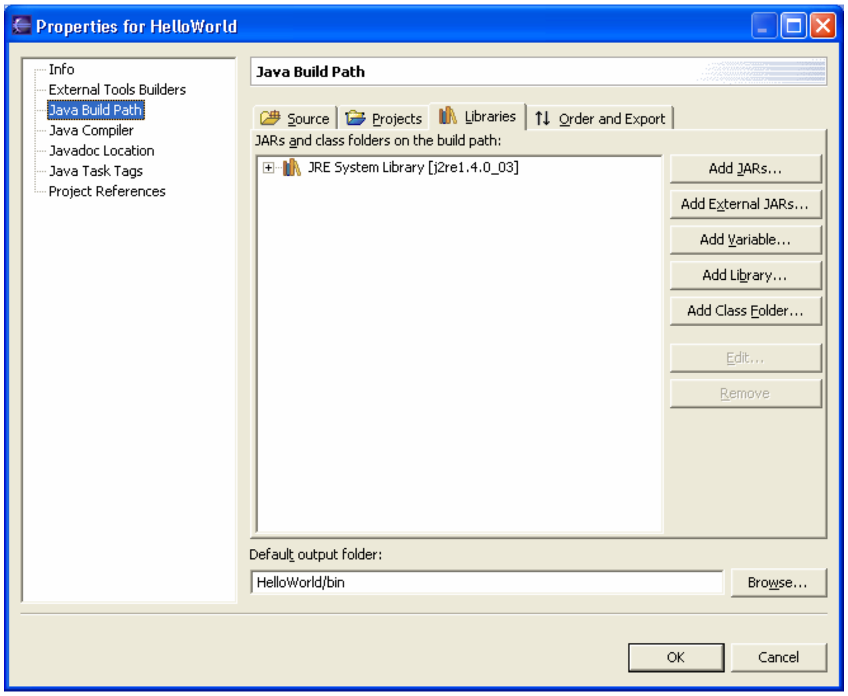


```
*/
public class SWTHello {

ublic static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    Label label = new Label(shell, SWT.NONE);
    label.setText("Hello, World!");
    shell.pack();
    label.pack();
    shell.open();
    while(!shell.isDisposed())
        if (!display.readAndDispatch())
            display.sleep();
    display.dispose();
    label.dispose();
```

9 items)

| Description | Resource | In Folder | Location |
|---|---|---|---|
| The import org.eclipse cannot be resol... | SWTHello.j... | HelloWorld/src | line 1 |
| The import org.eclipse cannot be resol... | SWTHello.j... | HelloWorld/src | line 2 |

In the tree-view choose Java Build Path. Then select the Libraries tab on the right side of the screen. Currently the only library listed is Sun's Java Runtime libraries necessary for any java program to execute. We now add swt.jar to this list. Select the "Add External JARs…" button on the right side of the screen.



JARs and class folders on the build path:

JRE System Library [j2re1.4.0_03]

Default output folder:

HelloWorld/bin

This will bring up a standard file selection dialog, and you must now find swt.jar. Its location is dependent on your platform. The SWT FAQ (available from the Eclipse website) gives the following list of locations for various platforms:

**win32:** INSTALLDIR\eclipse\plugins\org.eclipse.swt.win32_2.1.0\ws\win32\

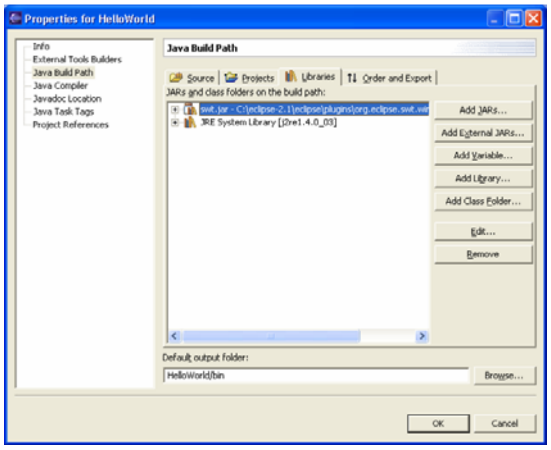
**gtk:** INSTALLDIR/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/ws/gtk/

**motif:** INSTALLDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.0/ws/motif/

**photon:** INSTALLDIR/eclipse/plugins/org.eclipse.swt.photon_2.1.0/ws/photon/

**macosx:** INSTALLDIR/eclipse/plugins/org.eclipse.swt.carbon_2.1.0/ws/carbon/

Once you have found and selected swt.jar, select Open. The swt.jar file should now be displayed in the list of libraries. When you hit OK and close the Properties dialog, the errors displayed earlier should be eliminated.

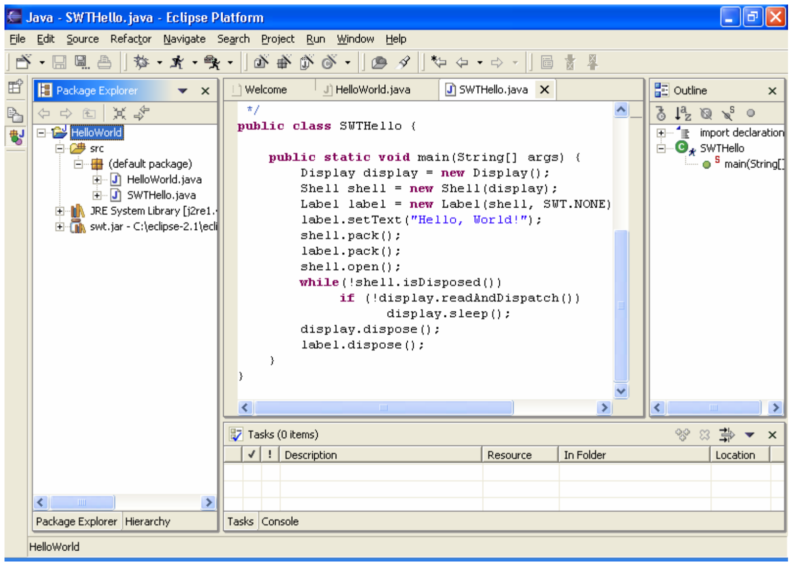


JARs and class folders on the build path:

```
swt.jar - C:\eclipse-2.1\eclipse\plugins\org.eclipse.swt.wi
JRE System Library [j2re1.4.0_03]
```

Default output folder: HelloWorld/bin

Click OK to close the window.



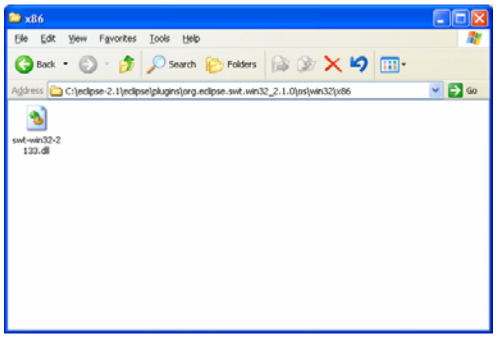
```
 */
public class SWTHello {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        Label label = new Label(shell, SWT.NONE)
        label.setText("Hello, World!");
        shell.pack();
        label.pack();
        shell.open();
        while(!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
        label.dispose();
    }
}
```

However, we are still not done. If you try running the program at this point, you will get an error similar to: "java.lang.UnsatisfiedLinkError: no swt-win32-2133 in java.library.path". The SWT run-time DLL that swt.jar requires is not available to the program. There are several ways of resolving this problem (described in the SWT FAQ also).

1. The first approach is to specify the location of the DLL as an argument to the Java virtual machine when you run the program. This is done by editing the launch configuration for your program. From the Run menu, choose "Run…", and select the specific launch configuration for your program in the tree-view on the left. Choose the Arguments tab on the right side of the screen. This allows you to set arguments to your program (if it takes input from the command-line) and to set arguments to the virtual machine that affect how it runs. In the text area labeled "VM arguments" enter the following line:

```
-Djava.library.path=<folder containing the SWT DLL>
```

As with swt.jar, the location of the DLL is platform dependent.



x86

Address: C:\eclipse-2.1\eclipse\plugins\org.eclipse.swt.win32_2.1.0\os\win32\x86

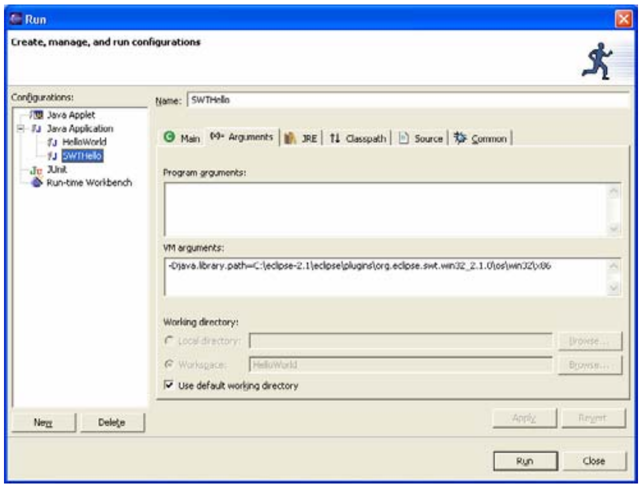
swt-win32-2133.dll

**Windows:** INSTALLDIR\eclipse\plugins\org.eclipse.swt.win32_2.1.0\os\win32\x86

**Linux GTK:** INSTALLDIR/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/os/linux/x86

**Linux Motif:** INSTALLDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.0/os/linux/x86

Note that if the path name contains spaces you must surround it with double-quotes, or you will get an error when launching the program.



Configurations: Java Applet, Java Application, HelloWorld, SWTHello, JUnit, Run-time Workbench

Name: SWTHello

Program arguments:

VM arguments: -Djava.library.path=C:\eclipse-2.1\eclipse\plugins\org.eclipse.swt.win32_2.1.0\os\win32\x86

Working directory:

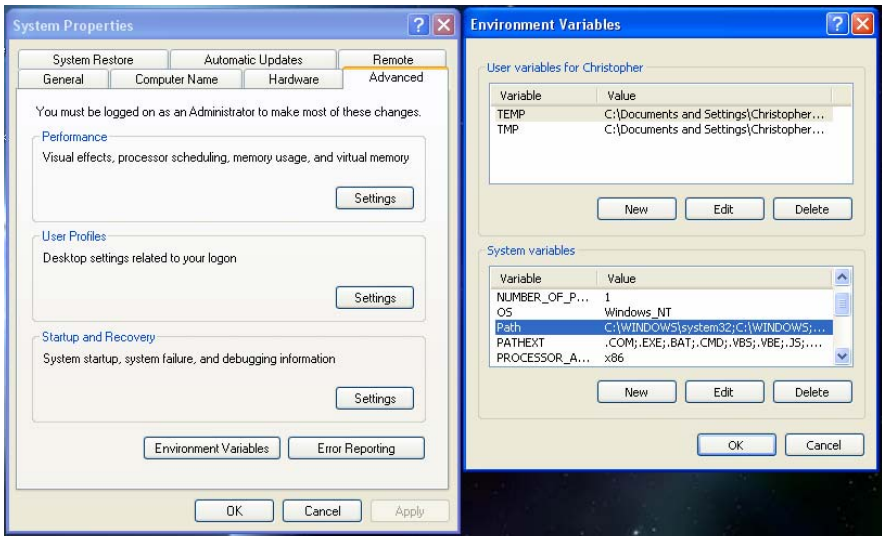
Local directory:

Workspace: HelloWorld

[✓] Use default working directory

2. The "-D" VM argument sets a system property, and in this case the property is the path where Java looks for libraries. A potential problem with the previous method is that you must remember to set this VM option every time you create a new launch configuration, which can be a nuisance. An alternative is to add the location of the DLL to the environment variable that Java uses for its library path. For Linux/UNIX modify LD_LIBRARY_PATH (as is described in more detail later in this document). For Windows, the PATH variable can be modified. On Win9X you can modify the PATH variable in the autoexec.bat file, and on NT/2K/XP it can

be modified through My Computer -> Properties -> Advanced -> Environment Variables.



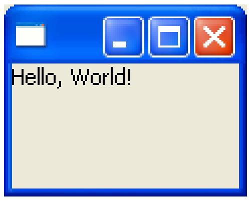
| Variable | Value |
|---|---|
| TEMP | C:\Documents and Settings\Christopher... |
| TMP | C:\Documents and Settings\Christopher... |

| Variable | Value |
|---|---|
| NUMBER_OF_P... | 1 |
| OS | Windows_NT |
| Path | C:\WINDOWS\system32;C:\WINDOWS;... |
| PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.... |
| PROCESSOR_A... | x86 |

3. A third option is to determine a location that is already specified by java.library.path, and copy the SWT DLL to that folder. A recommended standard choice (for Windows) is your Windows\System32 folder, but note that any location specified in the path environment variable discussed above will work. The disadvantage of this method is that if you upgrade Eclipse and get a different version of SWT, you must remember to copy the new SWT DLL to this location again.

4. The final option is to just copy the DLL into the (root of the) project folder. This has the drawback that you must do it for each project that uses SWT, and if you upgrade to a newer Eclipse/SWT version, you have to copy the file again.

Once you have selected and performed one of these options, the native code that enables the SWT to work will be available to the program, and it should run correctly. You should see a Shell (window), containing the text “Hello, World!” as shown below.

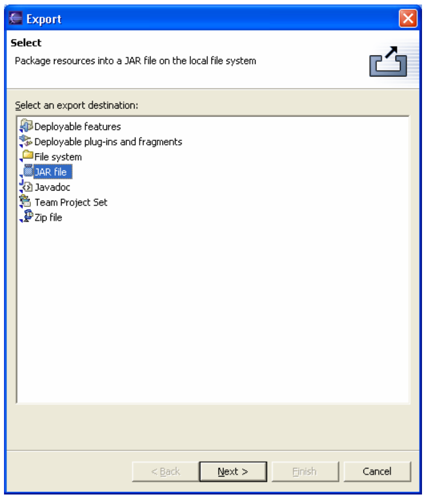


```
Hello, World!
```

Congratulations, you’ve just run your first SWT application!

## Creating a JAR for a stand-alone SWT application

Using the sample SWT application, we will now create an executable JAR file that can run from the command-line, outside of the Eclipse workbench. Right-click on the project name in the Package Explorer and choose the "Export" option. In the Export wizard page that appears, choose "JAR file" from the list, and click "Next".



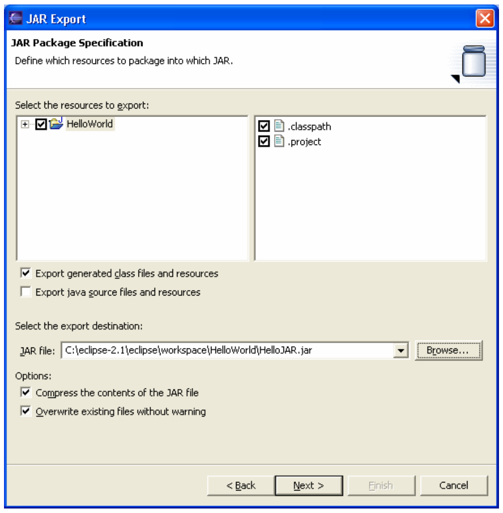
Select

Package resources into a JAR file on the local file system

Select an export destination:

- Deployable features
- Deployable plug-ins and fragments
- File system
- JAR file
- Javadoc
- Team Project Set
- Zip file

In the JAR Export screen that follows, use the tree-view at the top of the page to verify that the correct project and files are selected to be exported. Usually the defaults are correct. Check the "Export generated class files and resources" box, and if you wish to include the source in the JAR file, also check "Export java source files and resources." In the "JAR file:" text box, either type a location and name for the JAR file you would like to produce, or use the "Browse" button to select one using a file dialog. You can specify whether to compress the files and whether to automatically over-write files using the check boxes at the bottom.



JAR Package Specification

Define which resources to package into which JAR.

Select the resources to export:

- [✓] HelloWorld
- [✓] .classpath
- [✓] .project

- [✓] Export generated class files and resources
- [ ] Export java source files and resources

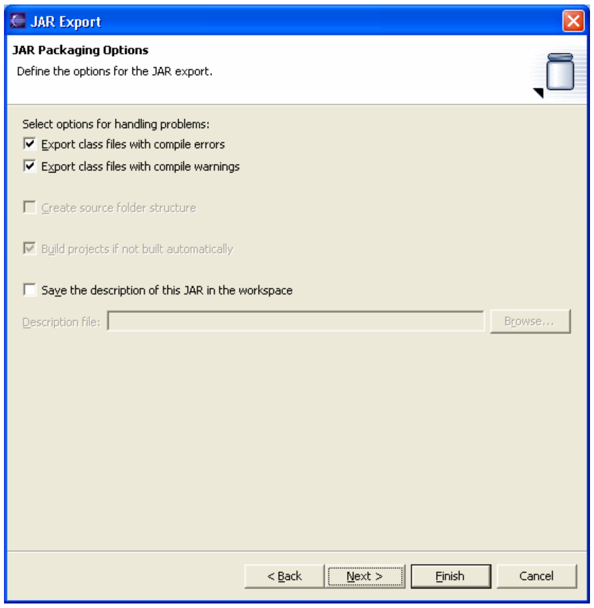
Select the export destination:

JAR file: C:\eclipse-2.1\eclipse\workspace\HelloWorld\HelloJAR.jar

Options:

- [✓] Compress the contents of the JAR file
- [✓] Overwrite existing files without warning

When you have completed this page, choose "Next." (Not Finish!)


JAR Export

**JAR Packaging Options**

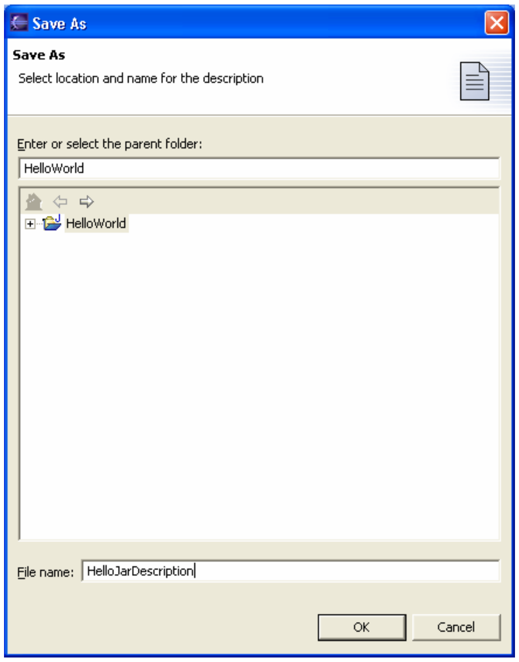
Define the options for the JAR export.

Select options for handling problems:

- [✓] Export class files with compile errors
- [✓] Export class files with compile warnings
- [ ] Create source folder structure
- [✓] Build projects if not built automatically
- [ ] Save the description of this JAR in the workspace

Description file:

On the page shown above, the default options will be fine, with the exception that you should check "Save the description of the JAR in the workspace." This way if you later make changes and wish to export the JAR again, you do not need to go through the entire Export Wizard a second time. Hit the Browse button to choose a location and filename.


Save As

**Save As**

Select location and name for the description

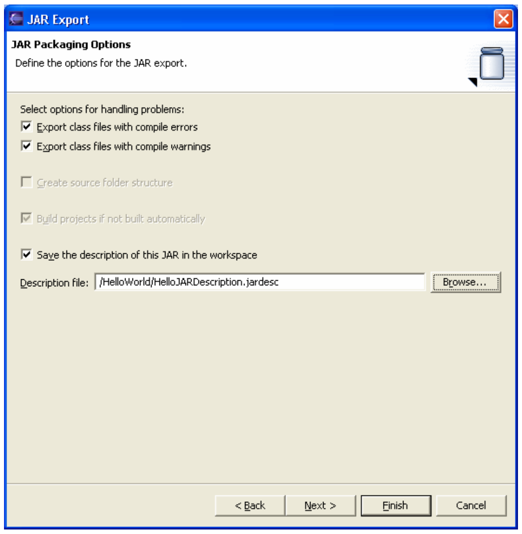
Enter or select the parent folder: HelloWorld

HelloWorld

File name: HelloJarDescription

Select the name of the project for which you are creating a JAR file and then enter a filename for the jar description file that will be created. (The suffix .jardesc will be appended to the name, unless you specify it yourself.)

The result should look like this.  Choose "Next"



Select options for handling problems:

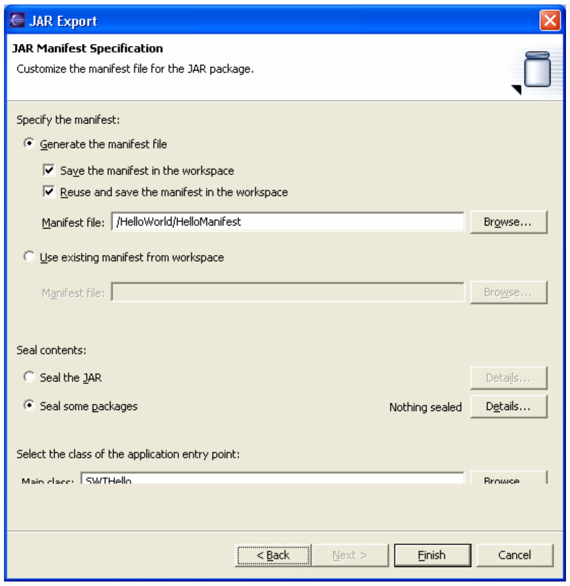
- [✓] Export class files with compile errors
- [✓] Export class files with compile warnings
- [ ] Create source folder structure
- [✓] Build projects if not built automatically
- [✓] Save the description of this JAR in the workspace

Description file: /HelloWorld/HelloJARDescription.jardesc

The final page of the JAR export Wizard allows you to set some of the available manifest options.  The manifest is a text file that is included in the JAR file, and which provides information about the JAR file.  For example, it can specify which is the Main class, what libraries the JAR depends on, whether the JAR is "sealed", and much more.  (Many of these options are not available from the Wizard.)  Select the "Generate the manifest file" radio button if it isn't already selected.  Check both "Save the manifest in the workspace" and "Reuse and save the manifest in the workspace".  These options together will allow you to make changes to the manifest, and have them reflected in the JAR file next time you generate it (using the JAR description file we asked to be generated on the previous page).  This will be useful in identifying the SWT libraries as necessary for running your JAR file.  Just as you did for specifying the jar description file, use the Browse button to select the project and enter a name for the manifest file.  In the example below, the manifest name is HelloManifest.

The Sun's java developer website explains package sealing as follows:  "A package within a JAR file can be optionally sealed, which means that all classes defined in that package must be archived in the same JAR file. You might want to seal a package, for example, to ensure version consistency among the classes in your software or as a security measure."  The options in the middle of the page allow you to seal some or all of the packages in your JAR file.

Lastly, the "Main class:" field specifies which class file contains the main method that will run your JAR file.  Use the Browse button to choose the main class from a list of available classes that have main() methods.


JAR Export

**JAR Manifest Specification**

Customize the manifest file for the JAR package.

Specify the manifest:

(●) Generate the manifest file

[✓] Save the manifest in the workspace

[✓] Reuse and save the manifest in the workspace

Manifest file: /HelloWorld/HelloManifest — Browse...

( ) Use existing manifest from workspace

Manifest file: — Browse...

Seal contents:

( ) Seal the JAR — Details...

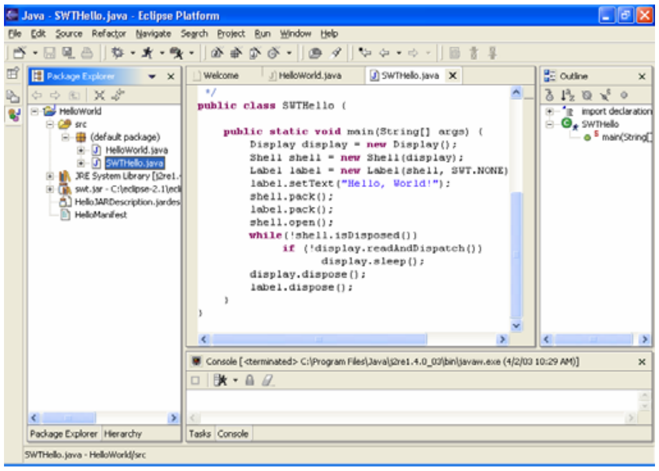
(●) Seal some packages — Nothing sealed — Details...

Select the class of the application entry point:

Main class: SWTHello — Browse

< Back | Next > | Finish | Cancel

If you have done all this, choose "Finish" to generate the JAR file (as well as the manifest and jar description files.)


Java - SWTHello.java - Eclipse Platform

Package Explorer: HelloWorld; src; (default package); HelloWorld.java; SWTHello.java; JRE System Library [j2re1.4...]; swt.jar - C:\eclipse-2.1\ecl...; HelloJARDescription.jardesc; HelloManifest

Tabs: Welcome | HelloWorld.java | SWTHello.java

```
 */
public class SWTHello {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        Label label = new Label(shell, SWT.NONE)
        label.setText("Hello, World!");
        shell.pack();
        label.pack();
        shell.open();
        while(!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
        label.dispose();
    }
}
```

Outline: import declarations; SWTHello; main(String[])

Console [<terminated> C:\Program Files\Java\j2re1.4.0_03\bin\javaw.exe (4/2/03 10:29 AM)]

SWTHello.java - HelloWorld/src

Both the JAR description file and manifest files should now be displayed in the Package Explorer view. The exported JAR file should be found in the target location you specified in the wizard. However, since the application also requires the SWT libraries to run, we must again take some additional steps to allow the new JAR file to use them. First, we must make the JRE aware of swt.jar. There are essentially two ways to do this. The first is to add swt.jar to your JRE\lib\ext folder. This folder is used for extensions/libraries for java, so the system knows where to look for them. These libraries will then be available at runtime for all your java programs. If you use this technique, ensure that you find the correct library: the JRE is typically installed in the Program Files folder. There may be another JRE folder inside the Java SDK (if it is installed) but this is not the correct location.

The second method is slightly more complex, but may be useful if you are distributing the program and cannot necessarily modify a user's JRE installation.

First, copy swt.jar to the same folder that contains the JAR file you just created. The manifest file must be modified to specify the fact that the JAR has a dependency on swt.jar, and cannot run without it. We do this by adding a line to the manifest file. (You can open it in Eclipse by double-clicking its name in the Package Explorer just like a java file.)

The line to add is: **`Class-Path: swt.jar`**

This will allow the new JAR file to use the classes in swt.jar. Now, right-click on the JAR description file in the Package Explorer and choose "Create JAR." If the JAR already exists (and it should if you've followed the steps above), it will prompt you to ensure that you want to overwrite the file. Choose Yes. When the jar file is distributed, the swt.jar file must be distributed with the jar and must be located in the same directory as the jar file.

Now, we need to make the SWT DLL available at run-time. Depending on the method you used to provide Eclipse with the location of the SWT DLL, you will need to perform similar steps now when running from the command-line.

Option 1) If you used the –Djava.library.path option, this option can also be specified when running the JAR from the command-line.

```
java –Djava.library.path="C:\eclipse\plugins\org.eclipse.swt.win32_2.1.0\os\win32\x86" –jar HelloJAR.jar
```

Options 2 or 3) If you added it to the PATH, then you are already done. The DLL is available everywhere.

Option 4) If you copied the DLL to the project folder, you must now copy the DLL to the same folder as your new JAR file.

You should now be able to execute the program from the command-line. Navigate to the correct folder, and typing the following command:

```
java –jar HelloJAR.jar
```

If your JAR file has a different name than HelloJAR.jar, substitute the name appropriately.

The output should appear identical to when you ran it from the Eclipse environment.

Hello, World!

When distributing the program, you must include both the application JAR file and the SWT jar file, and it is typically easiest to include the SWT DLL as well, rather than expecting the user to modify or know the PATH variable, or specify complicated command-line options.

## Importing an existing project

If you have been provided with an Eclipse project from another source, you can import it into your workspace and begin using it right away, rather than creating a new project from the beginning.

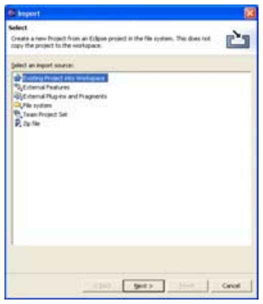
If you have been provided with the project in .zip format, first extract it to a location on your hard drive.  This will become the working directory for the project, so ensure that the location you choose is where you intend to store the project while you work on it.

For example, if you have been given a zip file called "SampleProject.zip", containing a project called "MyFirstProject" and you wish to place the project folder in "My Documents," extract the zip file into "My Documents", and the folder "MyFirstProject" will be created containing all the files for the project.  The folder name is always the same as the name of the project.

If you have been given the project uncompressed, simply copy the project folder (along with its contents) it to a location on your hard drive.

Ensure that the project folder contains a .project file.  This is an XML data file used by Eclipse to store information about the project.  It is used when Eclipse imports the project.

Start Eclipse, and from the file menu, choose the "Import" option.  The Import Wizard displays the following options.

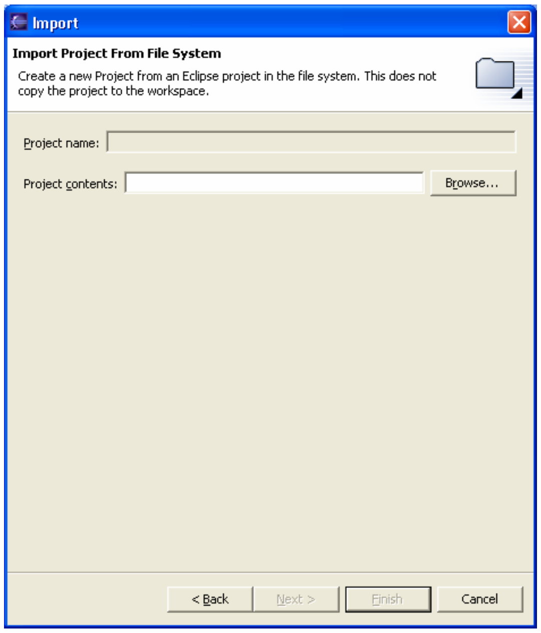

Import

Select

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Select an import source:

Choose “Existing Project into Workspace,” and hit “Next.” You must now specify the location of the project folder on your computer. Hit the browse button to bring up a “Browse for Folder” dialog. Select the project folder. The “Project contents:” text field should now display the location of the project folder, and the “Project name:” field should be filled in automatically with the project name, using information extracted from the “.project” file.



Import

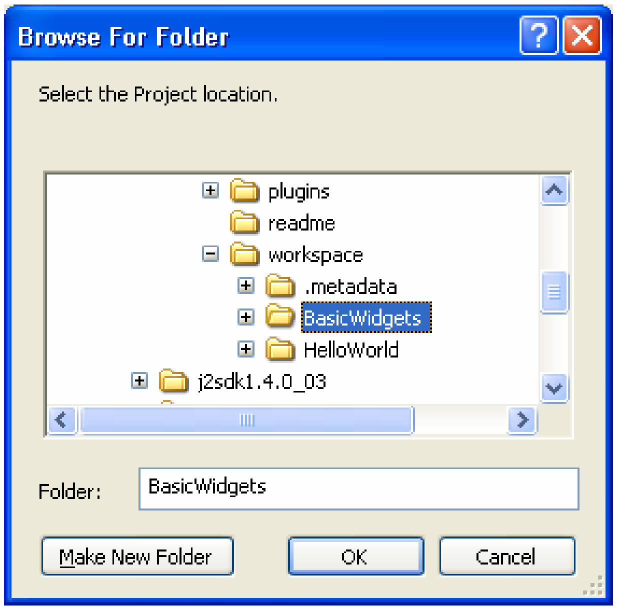
Import Project From File System

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Project name:

Project contents:

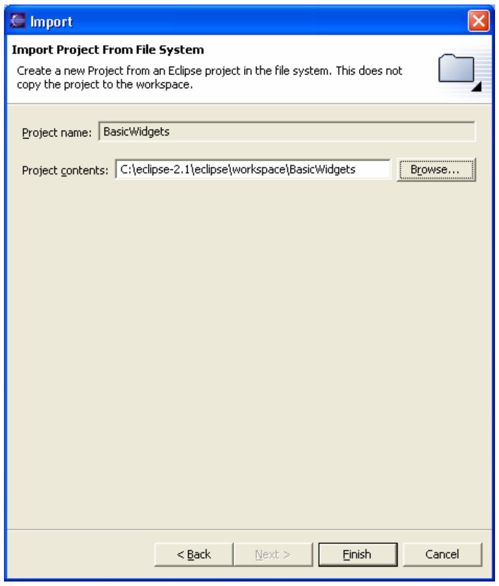
In this example, we copied a project titled BasicWidgets into the Eclipse workspace directory. We then navigated to the project and selected the project.



Browse For Folder

Select the Project location.

Folder: BasicWidgets


**Import**

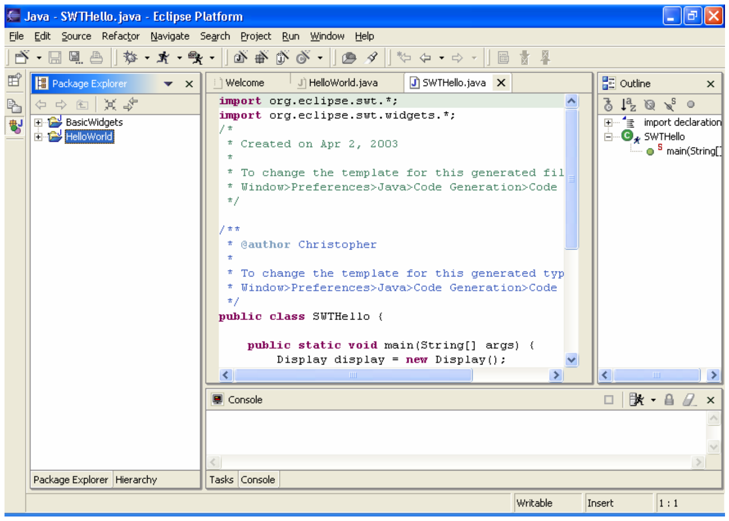
**Import Project From File System**

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Project name: BasicWidgets

Project contents: C:\eclipse-2.1\eclipse\workspace\BasicWidgets

When you select “Finish” the project is now displayed in the workspace.



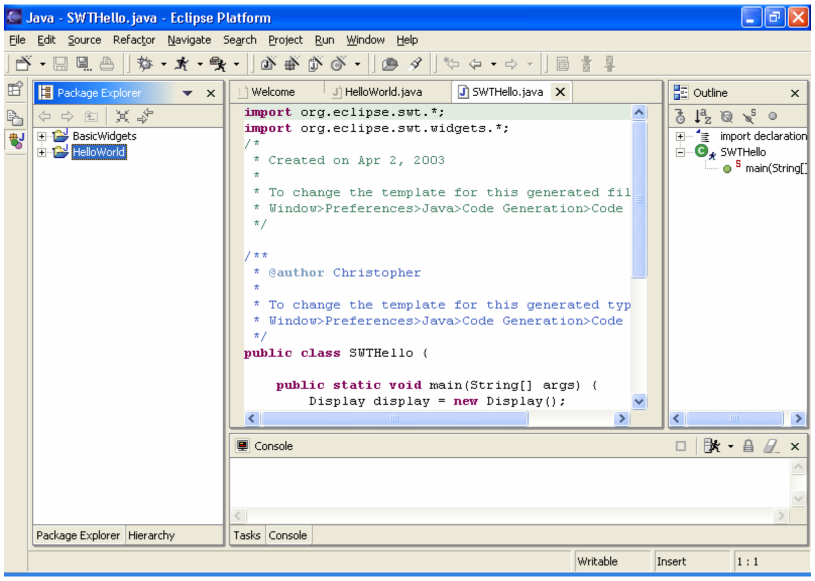
```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated fil
 * Window>Preferences>Java>Code Generation>Code
 */

/**
 * @author Christopher
 *
 * To change the template for this generated typ
 * Window>Preferences>Java>Code Generation>Code
 */
public class SWTHello {

    public static void main(String[] args) {
        Display display = new Display();
```

Note that if the project you imported employs the SWT user interface libraries, you will need to configure Eclipse to locate and use them correctly. This was covered in “Configuring Eclipse for programming with the SWT libraries”.

## Opening and Closing Projects

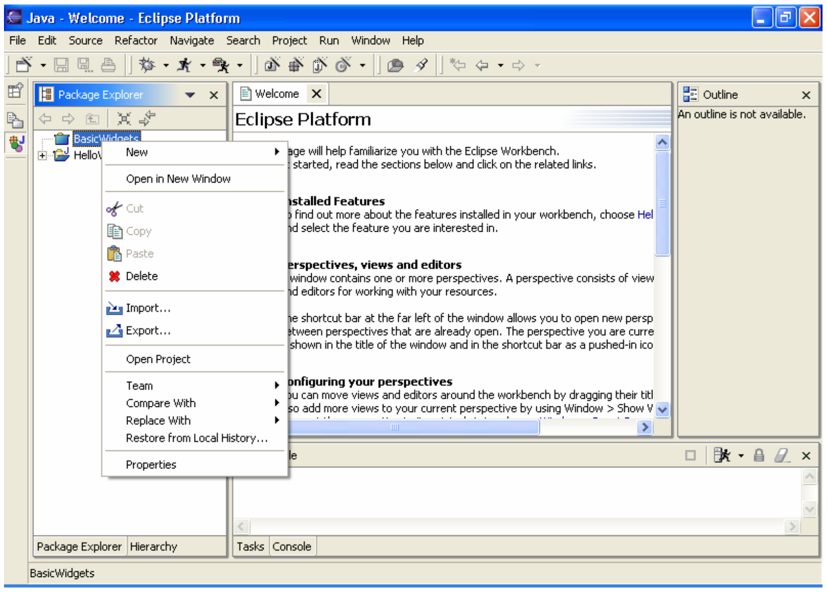
All projects that Eclipse is aware of are identified in the Package Explorer window. Projects that are “open” have a + beside the name, indicating that the project contents can be explored. Projects that are closed do not have a + beside the name. To open or close a project, select the project in the package explorer and then select Project → Open Project or Close Project.

In the workspace below, both the HelloWorld project and the BasicWidgets project are open and can be explored and run.



```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated fil
 * Window>Preferences>Java>Code Generation>Code
 */

/**
 * @author Christopher
 *
 * To change the template for this generated typ
 * Window>Preferences>Java>Code Generation>Code
 */
public class SWTHello {

    public static void main(String[] args) {
        Display display = new Display();
```

To delete a project, right-click on the project name and then select Delete.



Eclipse Platform

age will help familiarize you with the Eclipse Workbench.
started, read the sections below and click on the related links.

**nstalled Features**
find out more about the features installed in your workbench, choose Hel
d select the feature you are interested in.

**erspectives, views and editors**
window contains one or more perspectives. A perspective consists of view
d editors for working with your resources.

e shortcut bar at the far left of the window allows you to open new persp
tween perspectives that are already open. The perspective you are curre
shown in the title of the window and in the shortcut bar as a pushed-in ico

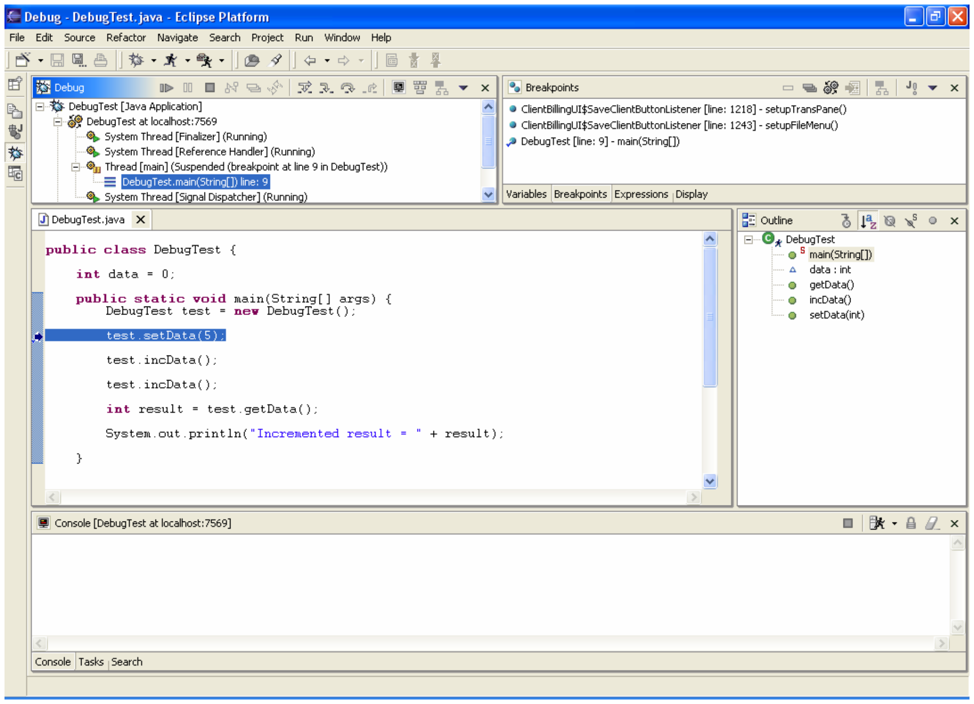
**onfiguring your perspectives**
u can move views and editors around the workbench by dragging their titl
so add more views to your current perspective by using Window > Show V

An outline is not available.

## The Debug Perspective

When developing an application, it is often useful to be able to step through the program line by line, and examine how the code is actually working. It often makes it possible to quickly find and remove subtle bugs that are more difficult to locate when running the program as a whole. The Java development tools that ship with Eclipse include a full-featured set of Java debugging tools.

These tools are usually located within the Debug Perspective, represented by the icon on the vertical toolbar on the left of the screen. An average debugging session will look something like the following, (assuming you haven't modified the default Debug perspective settings):


Debug - DebugTest.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Breakpoints
Variables Breakpoints Expressions Display
DebugTest.java
Outline
Console [DebugTest at localhost:7569]
Console Tasks Search

```
DebugTest [Java Application]
  DebugTest at localhost:7569
    System Thread [Finalizer] (Running)
    System Thread [Reference Handler] (Running)
    Thread [main] (Suspended (breakpoint at line 9 in DebugTest))
      DebugTest.main(String[]) line: 9
    System Thread [Signal Dispatcher] (Running)
```

```
ClientBillingUI$SaveClientButtonListener [line: 1218] - setupTransPane()
ClientBillingUI$SaveClientButtonListener [line: 1243] - setupFileMenu()
DebugTest [line: 9] - main(String[])
```

```java
public class DebugTest {

    int data = 0;

    public static void main(String[] args) {
        DebugTest test = new DebugTest();

        test.setData(5);

        test.incData();

        test.incData();

        int result = test.getData();

        System.out.println("Incremented result = " + result);

    }
```

```
DebugTest
  main(String[])
  data : int
  getData()
  incData()
  setData(int)
```

The Debug view in the top-left displays a stack-trace. It lists the existing threads and the method that each thread is currently in.

In the top-right there are four views stacked atop one another. The views are Variables, Breakpoints, Expressions and Display. The Variables view gives us access to a list of current variables and their values. The Breakpoints view lists the current breakpoints and their locations. The Expressions view allows us to enter "watch expressions" to be evaluated on the fly as we step through the code. Lastly, the Display view lets us enter code or expressions to be executed dynamically at the press of a button. We will return to look at these views in greater detail later in the tutorial.

The three views on the bottom two thirds of the screen are familiar to us from an earlier tutorial, displaying the code, an outline of the code, and the console output from the current program.

## Basic Debugging

To demonstrate how each component in the set of debugging tools works, we will look at a very simple Java class called DebugTest. It contains only one piece of data, and 3 methods to work with that data.

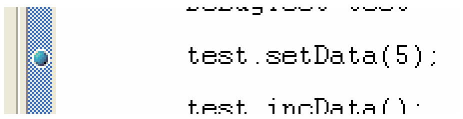
Create a new Java class with name DebugTest, and enter the following code for it:

```
public class DebugTest {
      int data = 0;

      public static void main(String[] args) {
            DebugTest test = new DebugTest();
            test.setData(5);
            test.incData();
            test.incData();
            int result = test.getData();
            System.out.println("Incremented result = " + result);

      }

      public void setData(int d) {
            data = d;
      }

      public void incData() {
            data++;
      }

      public int getData() {
            return data;
      }
}
```

To debug this program we must create a Launch Configuration as we did before when we simply wanted to run our programs. The only difference is that we execute it by choosing Debug… from the Run menu, or pressing the down arrow on the debug button, , and selecting the desired program. If you are not currently in the Debug Perspective, the default behaviour of Eclipse is to automatically switch to it. If you Debug right now, your program will run straight through to completion, without stopping. The result will be the following:

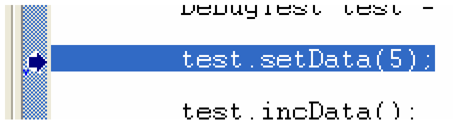
```
Incremented result = 7
```

This is not terribly useful to us, so what we must do is add "breakpoints" in our code. A breakpoint tells the Java debugger where we would like to halt execution of the code to take a closer look. There are several ways to create a breakpoint, the easiest of which is to click on the margin of the code editor beside the desired line of code. (The other two are pressing Ctrl-Shift-B, or choosing Add/Remove Breakpoint from the run menu, when the keyboard cursor is on the relevant line.)



```
test.setData(5);
```

The blue circle that appears represents a breakpoint. If we Debug our program again (by pressing the bug icon or just F11), code execution will stop prior to execution of the selected line, and the line will be highlighted blue with an arrow in the margin.



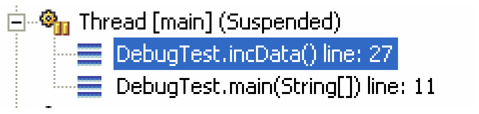
```
test.setData(5);

test.incData();
```

## Stepping Through Code

We can now step through the code one line at a time. We have several options, represented by buttons on the Debug view. We can "Step Over" a line of code with , "Step Into" a line of code with , or "Step Return" from a particular method using . These functions are also available from the Run menu.

"Step Over" means that if the line contains a method call, you will not enter that method but instead execute the current line entirely and pause on the next line of code in the current method. If the code is paused as in the above image, you can use the Step Over button to proceed to the next line, containing the command test.incData().

"Step Into" will do just the opposite, following execution into the method that has been called. It will pause on the first line of the called method. If execution is still paused on the first line containing test.incData(), pressing Step Into will pause on the first line of incData(), ie. data++. Notice that in the Debug view, the method incData() method is now listed on the stack trace.



```
Thread [main] (Suspended)
    DebugTest.incData() line: 27
    DebugTest.main(String[]) line: 11
```

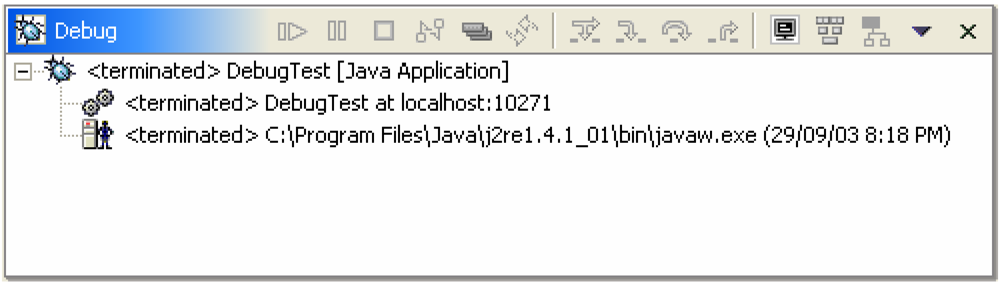
"Step Return" will skip execution ahead to the point where the current method has returned, and again pause. If you have followed the above steps and execution is still paused in the incData method, pressing Step Return will jump out of the method, and stop on the next line in the main method. incData() will correspondingly disappear from the stack trace in the Debug view.

(You may notice there is also a "Step with Filters/Step Debug" button, , which allows you to set filters to determine which methods your code should step into. For example, it would allow you to avoid stepping into 3rd party libraries, while still stepping through your own code. You can modify the filters using the Java->Debug->Step Filtering preferences page.)

To resume execution, we can press the Resume button represented by . Execution will continue normally, stopping again at any breakpoints that may occur later in the code.

To stop a program at any point in the middle of its execution, the Terminate button, , is used.

Once a program completes or is terminated, it will continue to be listed in the Debug view.



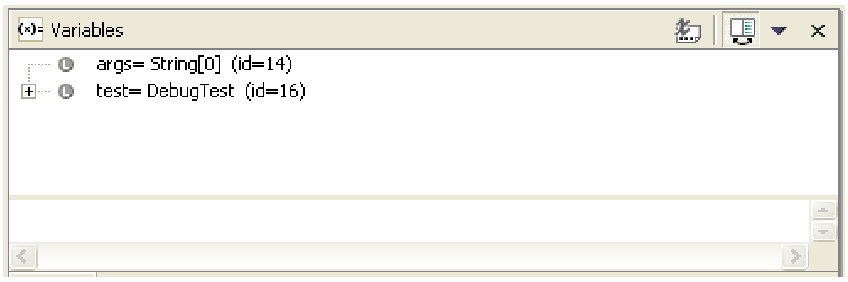
```
<terminated> DebugTest [Java Application]
    <terminated> DebugTest at localhost:10271
    <terminated> C:\Program Files\Java\j2re1.4.1_01\bin\javaw.exe (29/09/03 8:18 PM)
```

Even if we run another program, this old information will continue to be listed. To get rid of it, press the Remove All Terminated Launches button, shown as .

**Examining The Data**

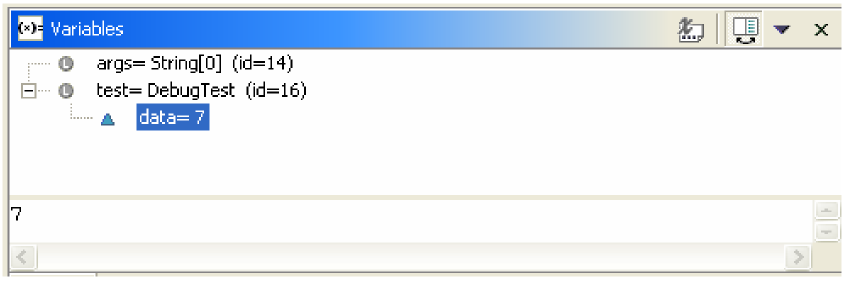
So far we have only looked at how to walk through the code, which may be useful in certain instances, but more often we would like to examine the state of the variables in our program as we proceed through it. To do this, we make use of the four views located in the top-right of the screen.

The Variables view shows us the state of all the local variables active in the current method. If we run the sample program and stop it at any point in the main method, we will see listed two variables, test and args. test is the instance of our main class, and args is the array of strings representing the input to the method from the command line.



```
args= String[0]  (id=14)
test= DebugTest  (id=16)
```

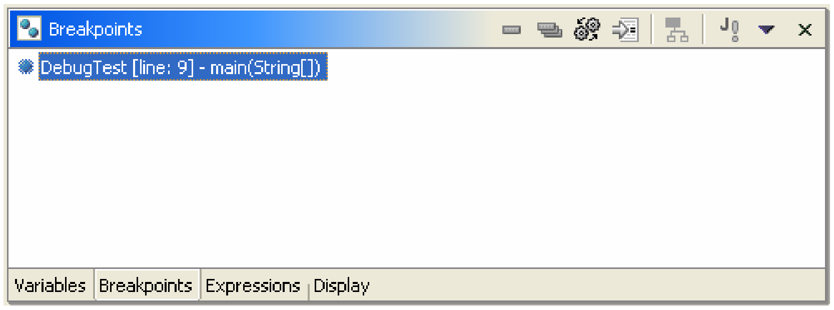
Since test is a sub-class of Object, we can examine the data members it contains by clicking on the + beside its name.  If any of the data members of test had been complex Objects, we could click the + beside their names, and continue recursively down the tree. An interesting example of this would be in debugging an implementation of a linked list, where each node contains references to other nodes.  To examine the current state of the list we could simply follow the links.



```
args= String[0]  (id=14)
test= DebugTest  (id=16)
    data= 7
7
```

The bottom section of the Variables view shows the value of the currently selected item. In this case, data has a value of 7 at the current location.

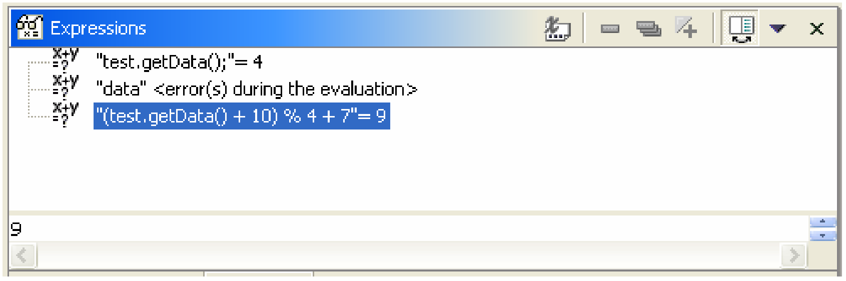
The next view is the Breakpoints view, which lists the current breakpoints.  By double-clicking on a breakpoint in the list you can see that line displayed in the Java editor.



```
DebugTest [line: 9] - main(String[])
```

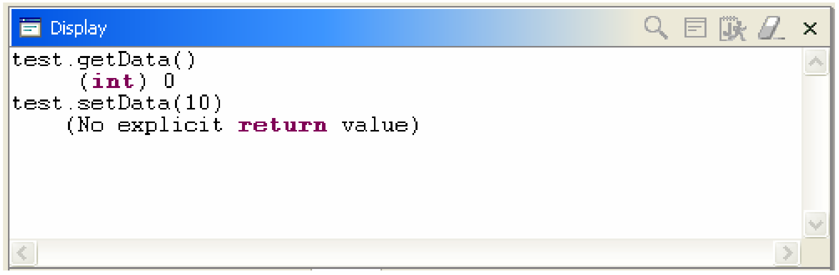
Another useful capability of the Breakpoint view is setting “hit counts” for breakpoints. If you right-click on a breakpoint in the list, and choose Hit Count you are prompted to enter a value.  This value will determine how many times execution must pass that breakpoint before stopping.  This is for debugging things like loops, in which you may only wish to see what is happening on the 421st iteration, for example.

The Expressions view allows you to specify particular expressions to evaluate on each step in order to see what is happening in your code. The expression may just be the name of a variable, or it can be a complex arithmetic expression. It can even contain method calls, like test.getData(). To add an expression, right-click on the view and choose "Add Java Watch Expression." Enter an expression, and it will appear in the list along with its current value. If a particular variable is not in scope, its value will be displayed as "<error(s) during the evaluation>."



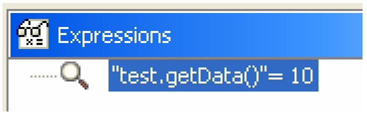
```
"test.getData();"= 4
"data" <error(s) during the evaluation>
"(test.getData() + 10) % 4 + 7"= 9
9
```

Closely related to the Expressions view is the Display view. It allows you to enter an expression or statement directly into it and run/evaluate it on command. To do so, simply click in the Display view, and type the expression or statement. Then select the entire expression and press the "Display Result of Evaluating Selected Text" button, , on the Display view. This will run the code, and display the return value (if there is one).



```
test.getData()
     (int) 0
test.setData(10)
    (No explicit return value)
```

Alternatively, if you press the left-most button ("Inspect Result…", ), the current value of the expression will be displayed in the Expressions view.



```
"test.getData()"= 10
```

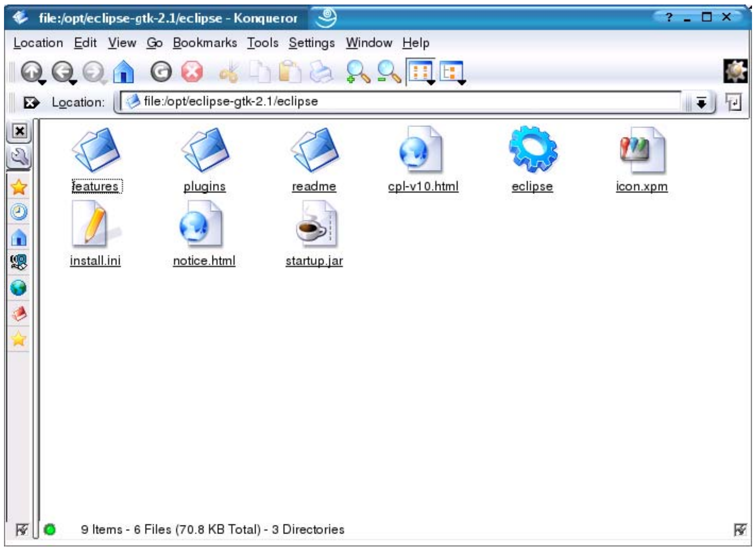
By right-clicking on it and choosing "Convert to Watch Expression" it will become a new watch expression and get updated as you continue through the code.

That rounds out the list of basic debugging facilities offered by Eclipse's Java Development tools. Although they are very straightforward, they should simplify a great deal of the debugging tasks that an average developer is faced with.

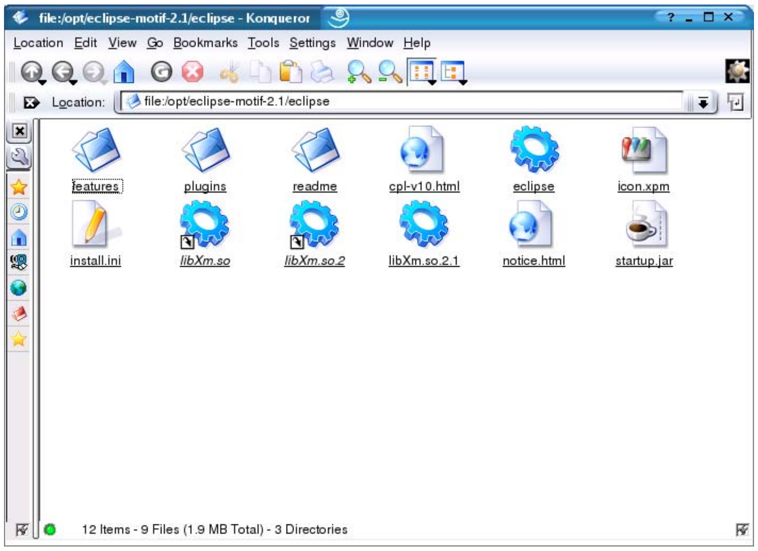
## Eclipse and Linux

Installing Eclipse under Linux is almost identical to the Windows install. In this section, we examine the differences between the Windows and the Linux installs.

In the following examples, Eclipse 2.1 was installed and run on SuSE Linux 8.1 (Office Desktop version) running the 2.4.19 kernel. Eclipse was installed in /opt to make it available to all users of the machine. The Java JRE 1.4.1 was used to run the programs. After downloading and unzipping the GTK version of Eclipse, the Eclipse folder has the following contents.


file:/opt/eclipse-gtk-2.1/eclipse - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: file:/opt/eclipse-gtk-2.1/eclipse
features
plugins
readme
cpl-v10.html
eclipse
icon.xpm
install.ini
notice.html
startup.jar
9 Items - 6 Files (70.8 KB Total) - 3 Directories

After installing the Motif version of Eclipse, the folder has the following contents.


file:/opt/eclipse-motif-2.1/eclipse - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: file:/opt/eclipse-motif-2.1/eclipse
features
plugins
readme
cpl-v10.html
eclipse
icon.xpm
install.ini
libXm.so
libXm.so.2
libXm.so.2.1
notice.html
startup.jar
12 Items - 9 Files (1.9 MB Total) - 3 Directories

If Eclipse is installed in a public directory, such as /opt, ensure that users have the appropriate permissions to run the system.

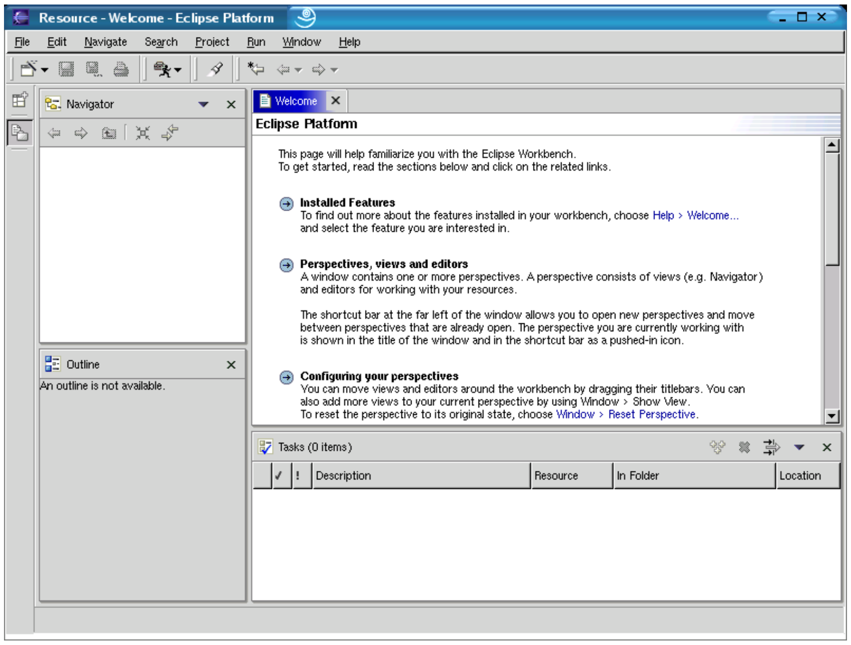
The following script can be used to run Eclipse.

```
export LD_LIBRARY_PATH=/opt/eclipse-gtk-2.1/eclipse:$LD_LIBRARY_PATH
cd /opt/eclipse-gtk-2.1/eclipse
./eclipse -ws gtk
```

The workspace is placed either in the Eclipse directory or in /home/*username*/workspace directory (it appears to depend on the permissions defined for the Eclipse directory). If necessary, the –data parameter can be included to indicate that the workspace is to be placed in a different directory.

```
export LD_LIBRARY_PATH=/opt/eclipse-gtk-2.1/eclipse:$LD_LIBRARY_PATH
cd /opt/eclipse-gtk-2.1/eclipse
./eclipse -ws gtk -data /home/user/gtk/workspace
```

When Eclipse with GTK is first run, the Welcome screen is displayed.



Eclipse Platform

This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related links.

**Installed Features**
To find out more about the features installed in your workbench, choose Help > Welcome... and select the feature you are interested in.

**Perspectives, views and editors**
A window contains one or more perspectives. A perspective consists of views (e.g. Navigator) and editors for working with your resources.

The shortcut bar at the far left of the window allows you to open new perspectives and move between perspectives that are already open. The perspective you are currently working with is shown in the title of the window and in the shortcut bar as a pushed-in icon.

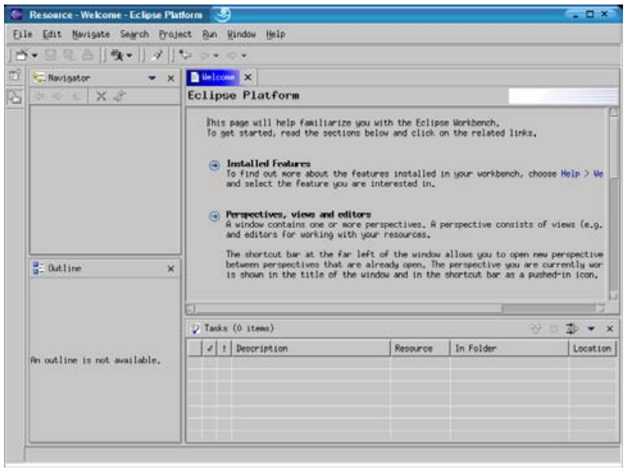
**Configuring your perspectives**
You can move views and editors around the workbench by dragging their titlebars. You can also add more views to your current perspective by using Window > Show View.
To reset the perspective to its original state, choose Window > Reset Perspective.

An outline is not available.

The following script is used to run the Motif version of Eclipse.

```
export LD_LIBRARY_PATH=/opt/eclipse-motif-2.1/eclipse:$LD_LIBRARY_PATH
cd /opt/eclipse-motif-2.1/eclipse
./eclipse -ws motif
```

Similarly, when Eclipse with Motif is first run, the Welcome screen is displayed.



Eclipse Platform

This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related links.

**Installed Features**
To find out more about the features installed in your workbench, choose Help > We
and select the feature you are interested in.

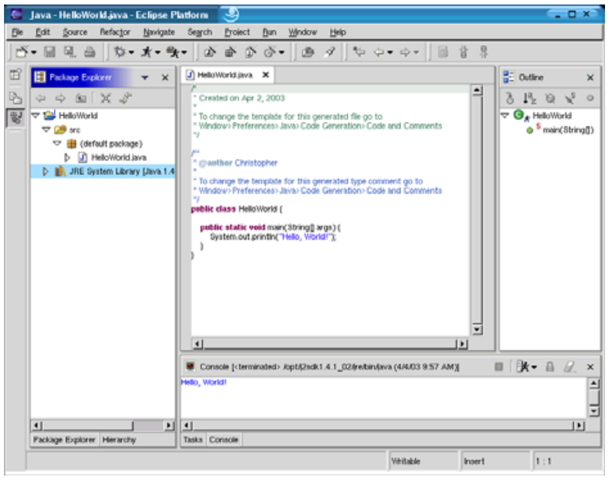
**Perspectives, views and editors**
A window contains one or more perspectives. A perspective consists of views (e.g.
and editors for working with your resources.

The shortcut bar at the far left of the window allows you to open new perspective
between perspectives that are already open. The perspective you are currently wor
is shown in the title of the window and in the shortcut bar as a pushed-in icon.

Tasks (0 items)

An outline is not available.

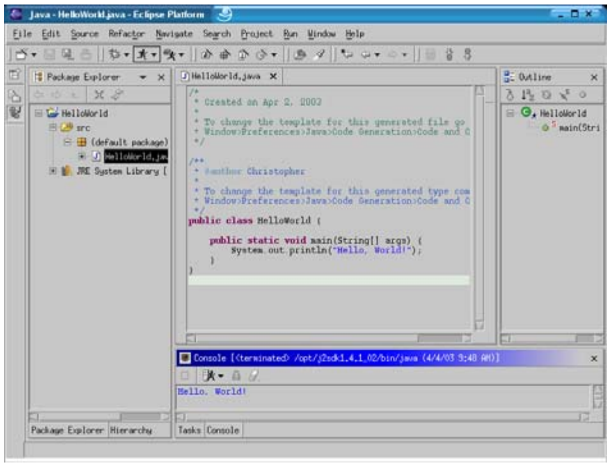
A new project is created in exactly the same manner as described earlier in the Windows section.  The simple HelloWorld program generates the following output on GTK.



```java
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated file go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */

/**
 * @author Christopher
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class HelloWorld {

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

```
Console [<terminated> /opt/j2sdk1.4.1_02/jre/bin/java (4/4/03 9:57 AM)]
Hello, World!
```

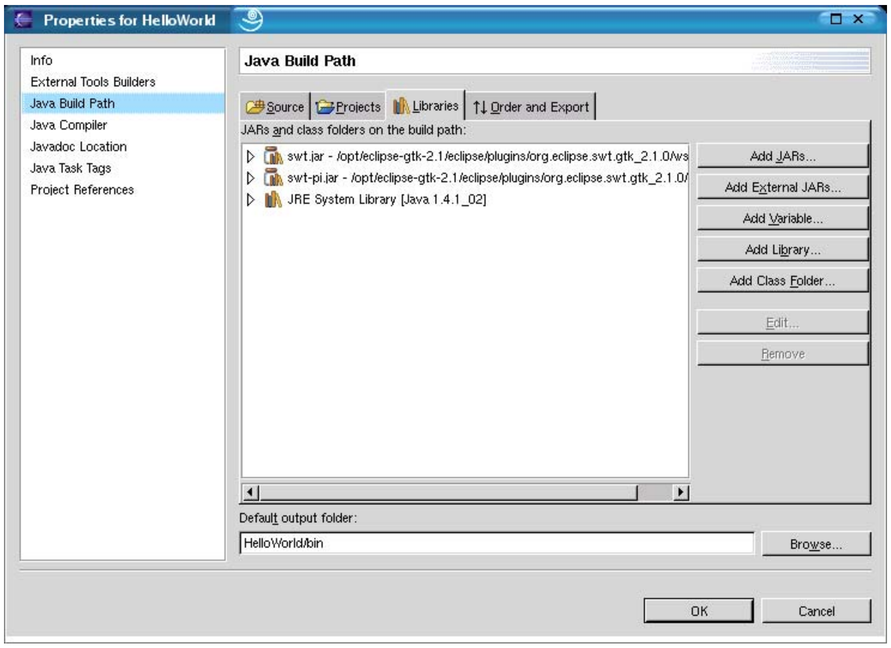
The simple HelloWorld program generates the following output on Motif.



```java
/*
 * Created on Apr 2, 2003
 *
 * To change the template for this generated file go
 * Window>Preferences>Java>Code Generation>Code and C
 */

/**
 * @author Christopher
 *
 * To change the template for this generated type com
 * Window>Preferences>Java>Code Generation>Code and C
 */
public class HelloWorld {

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

```
Console [<terminated> /opt/j2sdk1.4.1_02/bin/java (4/4/03 9:48 AM)]
Hello, World!
```

As noted in the Windows section, when SWT widgets are used in an application, the swt.jar library must be identified on the Java Build Path. As shown in the following screenshot, when SWT widgets are included in a GTK program, the swt-pi.jar file must also be included on the Java Build Path.

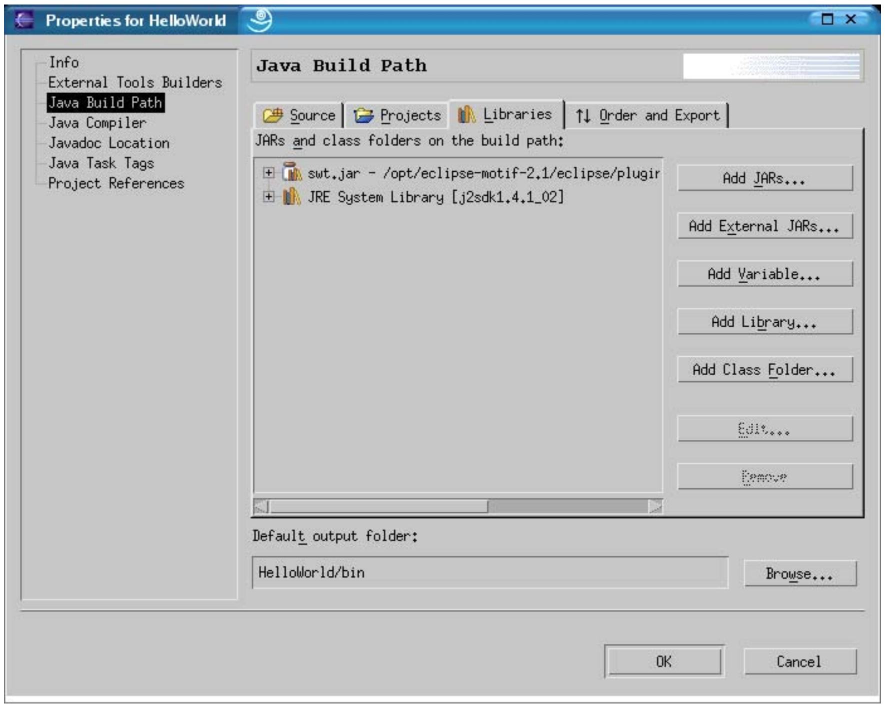


JARs and class folders on the build path:

swt.jar - /opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/ws
swt-pi.jar - /opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/
JRE System Library [Java 1.4.1_02]

Default output folder: HelloWorld/bin

Similarly, when Motif is used, the swt.jar file must be included.



JARs and class folders on the build path:

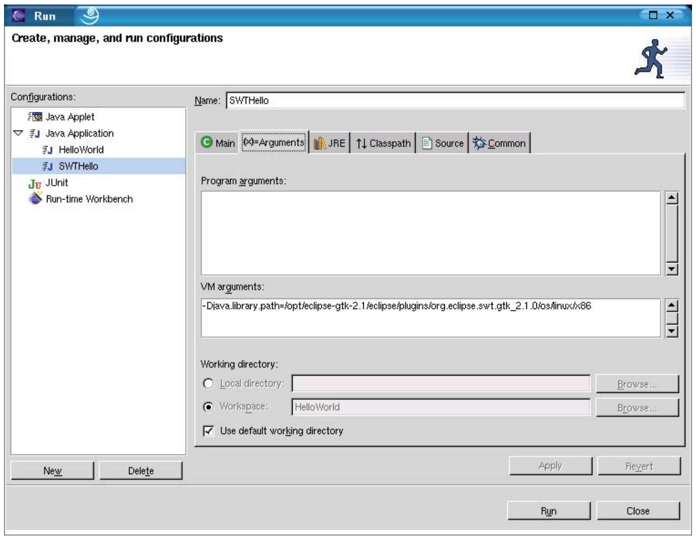
swt.jar - /opt/eclipse-motif-2.1/eclipse/plugir
JRE System Library [j2sdk1.4.1_02]

Default output folder: HelloWorld/bin

When SWT widgets are included in an application, the appropriate run-time libraries must be identified to Eclipse; if these libraries are not available, a run-time error is generated.

The run-time libraries are identified in the VM arguments section of the Run command:

```
-Djava.library.path=/opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/os/linux/x86
```



Name: SWTHello

VM arguments: -Djava.library.path=/opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/os/linux/x86

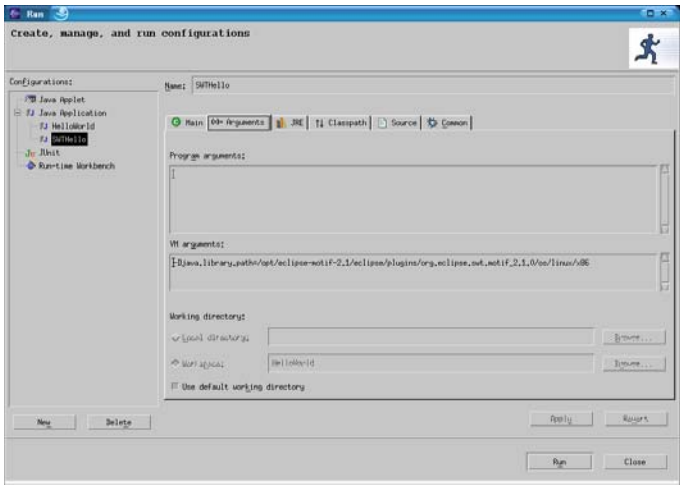
Local directory:

Workspace: HelloWorld

Use default working directory [✓]

The Motif run-time libraries are identified in the same manner:

```
-Djava.library.path=/opt/eclipse-motif-2.1/eclipse/plugins/org.eclipse.swt.motif_2.1.0/os/linux/x86
```



Name: SWTHello

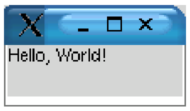
VM arguments: -Djava.library.path=/opt/eclipse-motif-2.1/eclipse/plugins/org.eclipse.swt.motif_2.1.0/os/linux/x86

Local directory:

Workspace: HelloWorld

Use default working directory

Finally, once everything has been defined, the Hello World program runs correctly.



Hello, World!

## Creating and running a jar file

Generating an executable jar file for a Linux SWT application is almost identical to the techniques described in the Windows section. In this section, we identify the differences that exist between Linux and Windows.

As was mentioned in the Windows section, the swt.jar file (for GTK, the swt.jar and swt-pi.jar files) must be available at run-time. These files may be made available by copying them to the Sun's Java jre/lib/ext directory. Alternatively, the two files could be placed in the directory that will contain the jar file.

Use the Export command as was described in the Windows section. If the swt.jar file is to be placed in the Java ext directory, the following manifest file may be used.

```
Manifest-Version: 1.0
Main-Class: SWTHello
```

If the swt.jar file is to be included in the same directory as the SWT application, the following manifest file may be used. Note that the swt-pi.jar is required only for the GTK window system. For this manifest file to work, the swt.jar (and swt-pi.jar) must be copied to the directory in which the jar file is stored.

```
Manifest-Version: 1.0
Main-Class: SWTHello
Class-Path:  swt.jar  swt-pi.jar
```

If it is preferred that the files not be copied to the jar directory, then the following manifest file may be used to specify the location of the jar files at run-time. Note that the file names may be split over multiple lines, simply ensure that there is a blank in column 1 of the continued line(s).

```
Manifest-Version: 1.0
Main-Class: SWTHello
Class-Path: file:/opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/ws/gtk/swt.jar
  file:/opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/ws/gtk/swt-pi.jar
```

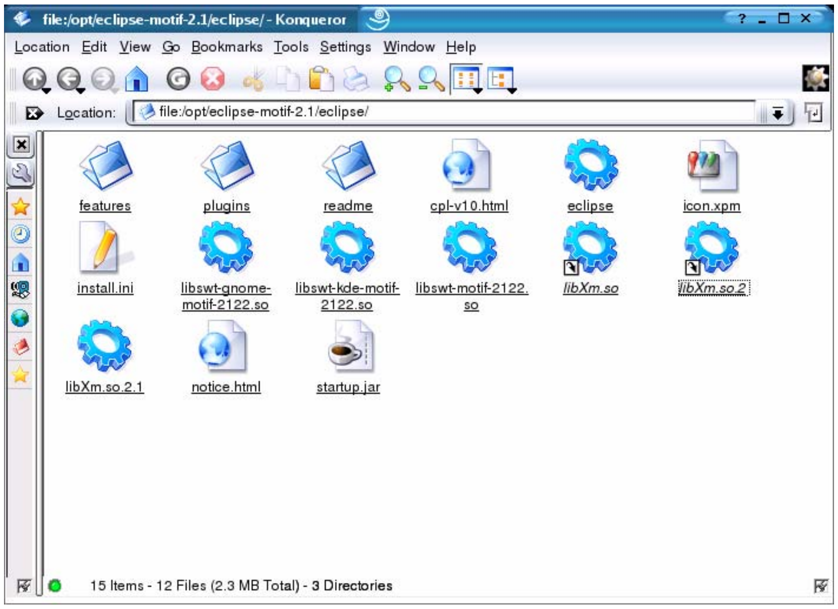
Once the jar file has been built, it may be run from the command prompt with the following script for GTK:

```
export LD_LIBRARY_PATH=/opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/os/linux/x86/:$LD_LIBRARY_PATH
cd workspace/HelloWorld
java -jar HelloWorld.jar
```

or with the following GTK script:

```
cd workspace/HelloWorld
java  -Djava.library.path=/opt/eclipse-gtk-2.1/eclipse/plugins/org.eclipse.swt.gtk_2.1.0/os/linux/x86  -
jar HelloWorld.jar
```

If you are running with Motif, two sets of run-time routines are required: those in the eclipse folder and those in the plugins os folder. To simplify running the jar, we copied the 3 run-time files from the plugins os folder to the top level Eclipse folder. The contents of the folder are shown below.

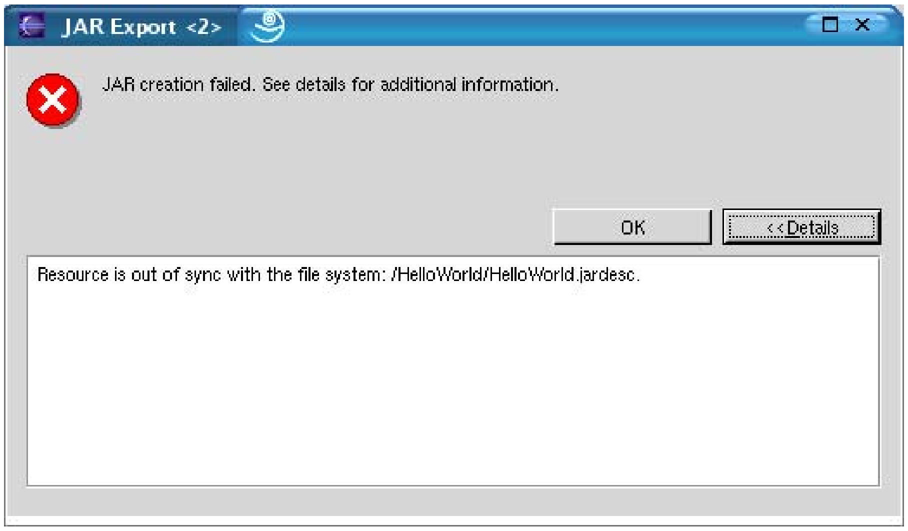

file:/opt/eclipse-motif-2.1/eclipse/ - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: file:/opt/eclipse-motif-2.1/eclipse/
features
plugins
readme
cpl-v10.html
eclipse
icon.xpm
install.ini
libswt-gnome-motif-2122.so
libswt-kde-motif-2122.so
libswt-motif-2122.so
libXm.so
libXm.so.2
libXm.so.2.1
notice.html
startup.jar
15 Items - 12 Files (2.3 MB Total) - 3 Directories

The Motif jar file can now be run with the following script:

```
export LD_LIBRARY_PATH=/opt/eclipse-motif-2.1/eclipse/:$LD_LIBRARY_PATH
cd workspace/HelloWorld
java -jar HelloWorld.jar
```

To avoid copying the run-time routines to the same directory, the export command that sets the library path could be expanded to include both libraries.

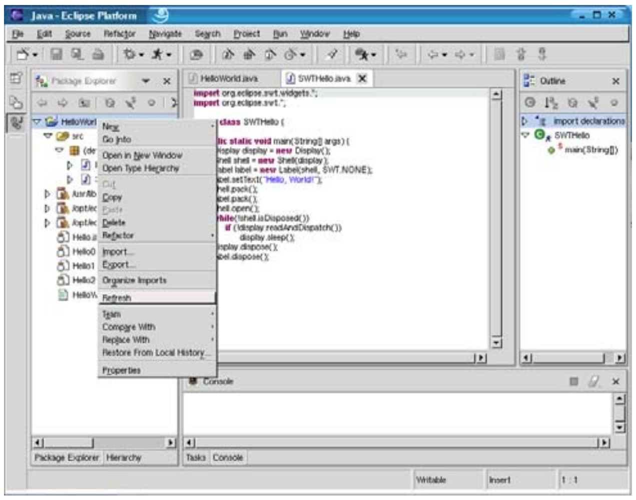
When you are building a jar file for the first time and making changes to the manifest file at the same time, you may receive a jar message that indicates the resources are not in sync.



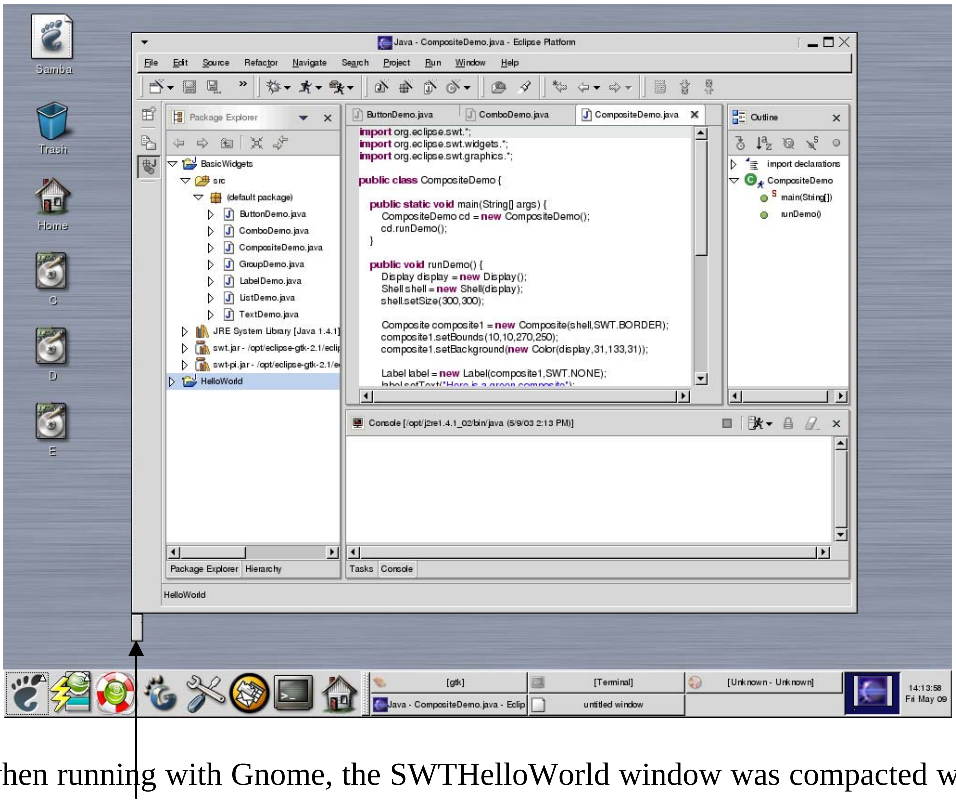
JAR creation failed. See details for additional information.

Resource is out of sync with the file system: /HelloWorld/HelloWorld.jardesc.

This message is not particularly self-explanatory but the problem can be fixed by refreshing the current project. To do this, right-click on the project name and select Refresh from the menu. This should eliminate the problem.



We encountered a few minor problems when running Eclipse with Linux. With SuSE Linux/KDE, the accelerator combination for Run Last Launched (cntl-F11) did not work, although it did work correctly on the same machine when running Windows XP and when running Linux/Gnome. The Debug Last Launched (F11) did work correctly on SuSE Linux/KDE.

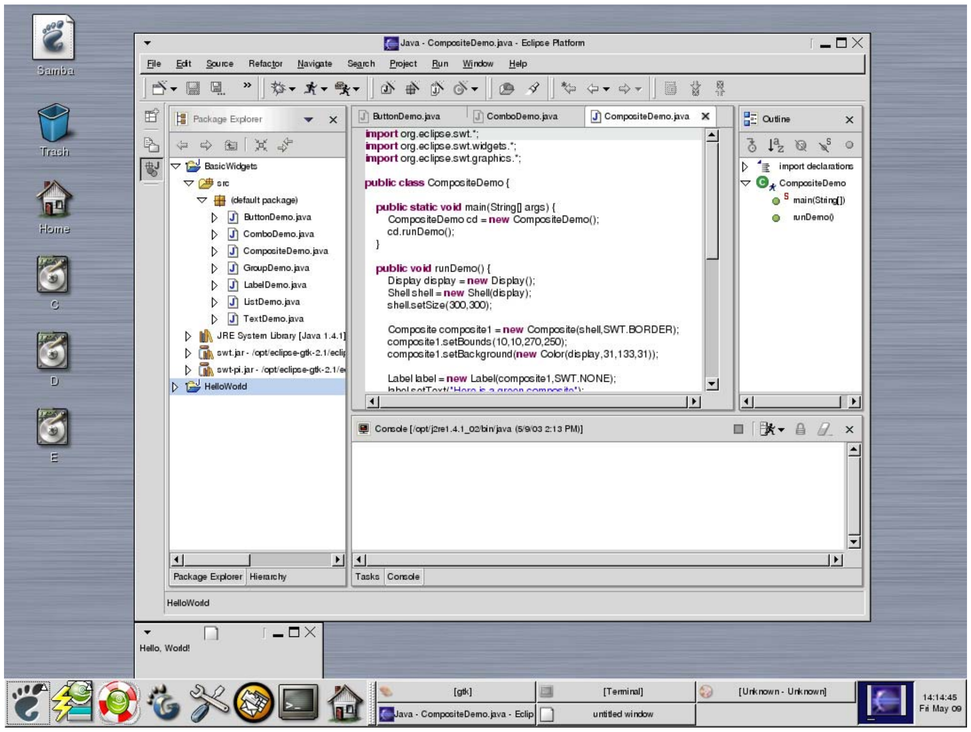


```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;

public class CompositeDemo {

  public static void main(String[] args) {
    CompositeDemo cd = new CompositeDemo();
    cd.runDemo();
  }

  public void runDemo() {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setSize(300,300);

    Composite composite1 = new Composite(shell,SWT.BORDER);
    composite1.setBounds(10,10,270,250);
    composite1.setBackground(new Color(display,31,133,31));

    Label label = new Label(composite1,SWT.NONE);
```

Console [/opt/j2re1.4.1_02/bin/java (5/9/03 2:13 PM)]

Also, when running with Gnome, the SWTHelloWorld window was compacted when the application was run and, as a result, it was easy to miss the window on the desktop.

Clicking on the bottom-right corner and dragging the window makes the window visible.



```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;

public class CompositeDemo {

  public static void main(String[] args) {
    CompositeDemo cd = new CompositeDemo();
    cd.runDemo();
  }

  public void runDemo() {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setSize(300,300);

    Composite composite1 = new Composite(shell,SWT.BORDER);
    composite1.setBounds(10,10,270,250);
    composite1.setBackground(new Color(display,31,133,31));

    Label label = new Label(composite1,SWT.NONE);
```

Hello, World!

## Summary

This document contains a brief introduction to installing and configuring Eclipse so that Java programs can use the SWT. We have attempted to keep the configuration details simple to avoid confusing the first-time user.

We have deliberately not included a comprehensive description of the Eclipse workbench. For more information on the workbench, a document titled The Eclipse Workbench User's Guide is available at:

**`http://eclipsewiki.swiki.net/239`**

The Eclipse Faq is currently available at:

**http://www.eclipse.org/eclipse/faq/eclipse-faq.html**

As you become more familiar with the workbench, you will likely want to use accelerator keys instead of the menus for frequently used operations. A list of the accelerator keys is available at:

**`http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-ui-home/accessibility/keys.html`**