Getting Started with Eclipse for Java and GridWorld

Maria Litvin
Phillips Academy, Andover, Massachusetts

Gary Litvin
Skylight Publishing

1. Introduction

Eclipse Software Development Kit (Eclipse SDK) is a vast extendable set of tools for software development. Here we are interested in Eclipse's Integrated Development Environment (IDE) component for writing Java software. Eclipse is an open source project of Eclipse Foundation; you can find information about Eclipse Project at http://www.eclipse.org/eclipse. Eclipse is available free of charge under the Eclipse Public License.

Eclipse was developed by software professionals for software professionals; it may seem overwhelming to a novice. This document describes the very basics of Eclipse, enough to get started with Java in an educational setting.

Eclipse runs on multiple platforms including Windows, Linux, and Mac OS. There may be minor differences between Eclipse versions for different platforms and operating systems, but the core features work the same way. Here we will use examples and screen shots from Windows.

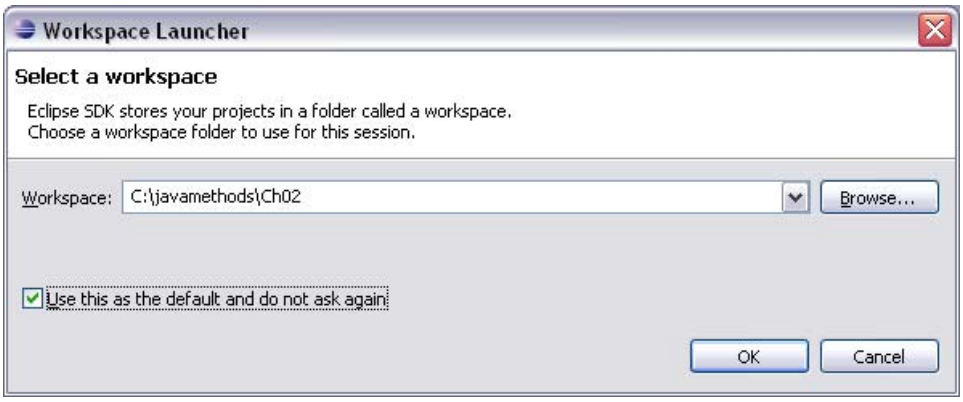
This document describes Version 3.6 of Eclipse, known as Eclipse Helios, released in June 2010.

2. Downloading and Installing Eclipse

Go to http://download.eclipse.org/eclipse/downloads and click on the link for the latest release (for example, `3.6`). Scroll down and find your operating system in the list, for example, Windows (Supported Versions), click on the `(http)` link in the "Download" column and download the compressed Eclipse file.

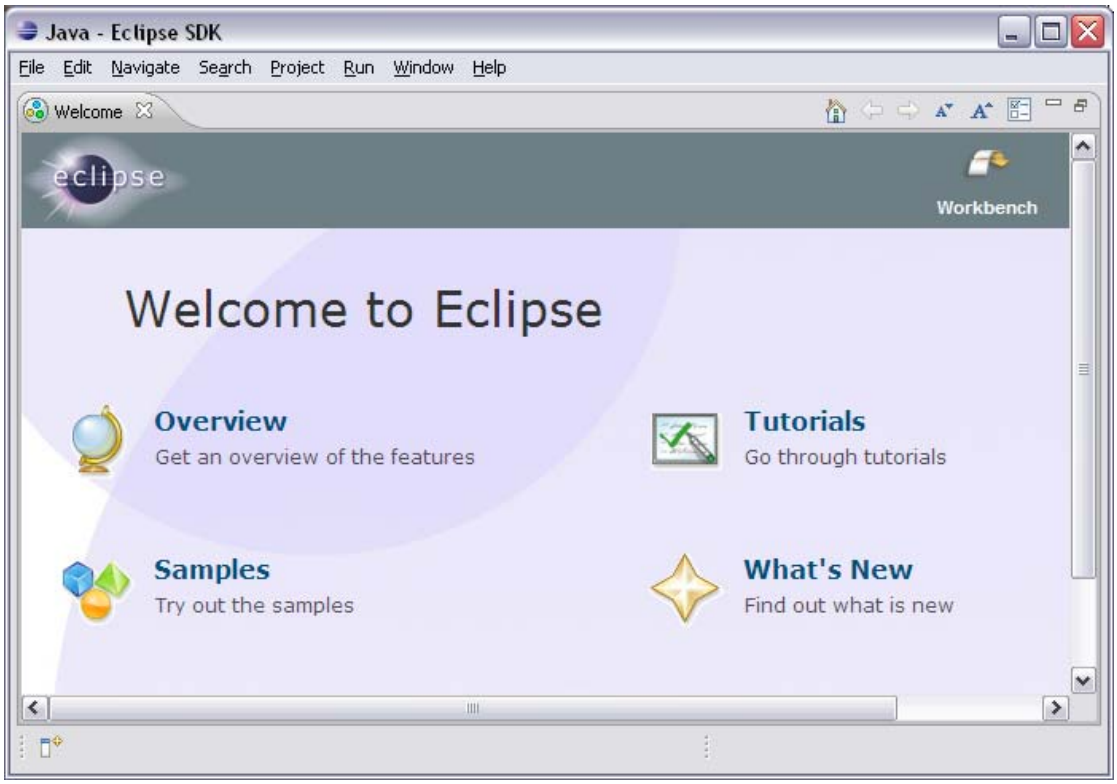
Under Windows, you will download a `zip` file, for example, `eclipse-SDK-3.6-win32.zip`. Double-click to open it and copy the `eclipse` folder from the zip file into your file system. (It may take a few minutes to extract Eclipse files from the zip file.) We prefer to put the `eclipse` folder into `C:\Program Files` and rename it `Eclipse` (with a capital "E") for consistency with the names of other application folders. But you can install it in a root directory, for example, `C:\eclipse`. You will find `eclipse.exe` in the `eclipse` folder. This is the Eclipse executable. Create a shortcut to it on the desktop (by dragging `eclipse.exe` to the desktop while holding down `Ctrl+Shift)`.

Double click on the shortcut or on `eclipse.exe`. A dialog box will pop up asking you to choose a workspace:



In Eclipse a workspace is basically a folder that contains project files and configuration data. Enter a folder of your choice, for example `C:\javamethods\Ch02`, for projects from Chapter 2 of *Java Methods*. Check the "Use this as the default" box — you can change it later. Click `OK`.

Eclipse then comes up with a Welcome screen:



Close it (click on the X on the "Welcome" tab) unless you want to go over the overview and/or tutorials.

3. Importing and Exporting the Preferences

Configuring Eclipse is a daunting and time-consuming task for a novice. Eclipse has thousands of configurable options, basic and advanced, all mixed together (for example, "Insert spaces for tabs" and "Show affordance in hover on how to make it sticky" in the same dialog). Factory defaults are chosen for experienced software developers and are not always appropriate for educational use.

Eclipse configuration settings apply only to the current workspace and revert to defaults when you switch to a new workspace. Theoretically it is possible to always use the same workspace, but as the number of projects in it grows, it may become unmanageable and the Eclipse performance may be degraded. We prefer to use a separate workspace for each chapter in the book. Luckily, Eclipse provides a way to export the preferences from the current workspace into a file (an `.epf` file) and import the preferences from a file into a workspace.

Our preferences are available in the `LitvinsPreferences.epf` file. You can download `LitvinsPreferences.zip`, which contains `LitvinsPreferences.epf`, from http://www.skylit.com/javamethods/faqs/LitvinsPreferences.zip.

If you want, just import these preferences into your workspace and leave Section 4, "Configuring Eclipse," until later, when you are ready to experiment with your own settings. Follow these steps:

1. Download `LitvinsPreferences.epf` into a folder of your choice.
2. Choose the `Import...` command on the `File` menu.
3. Expand "General", select "Preferences" and click `Next`:

4. Browse to `LitvinsPreferences.epf` and click `Finish`.

You need to configure or import preferences into every workspace that you create.

If you want to save your preferences, perhaps for backup or for using them in another workspace, export them into a file. Follow these steps:

1. Choose the `Export...` command on the `File` menu.
2. Expand "General", select "Preferences" and click `Next`.
3. Type the pathname of the file (or browse to the folder where you want to store the `.epf` file and add the file name). No need to include the `.epf` extension. Click `Finish`.

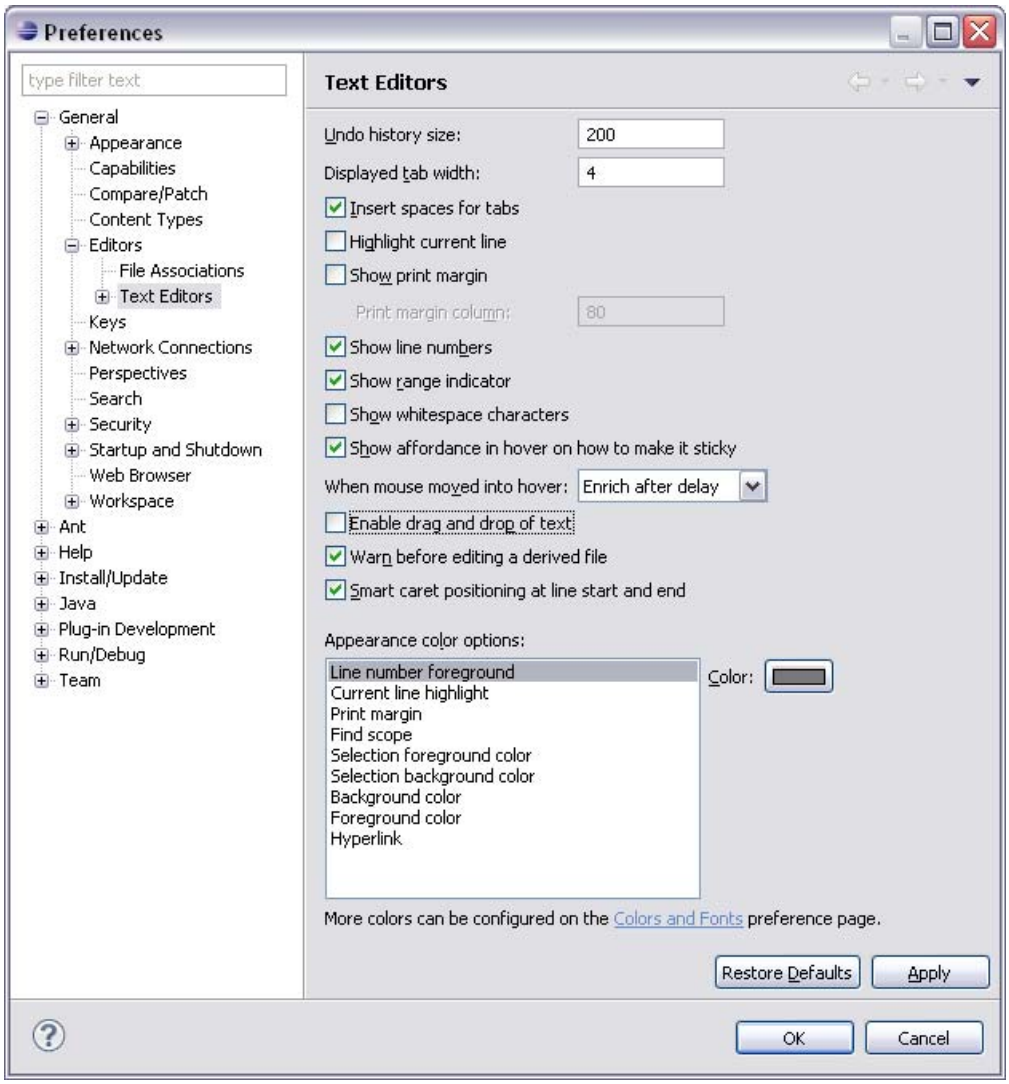
4. Configuring Eclipse

In this section we give a few suggestions for setting preferences and show where different types of options are located. The options we recommend below simply reflect... well, our preferences.

In Eclipse, the `Preferences` command is located under the `Window` menu.

Click on it.

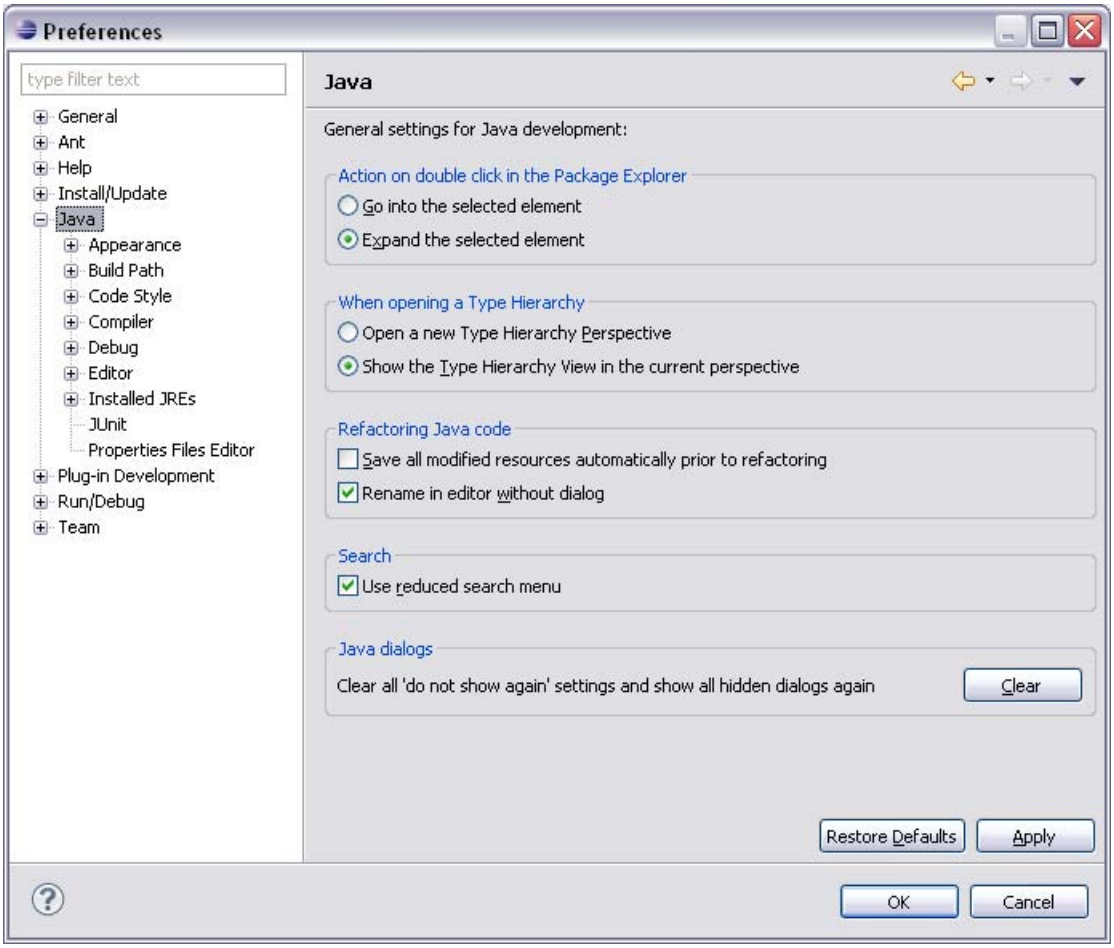
Under **General⇨Editors⇨Text Editors** check "Insert spaces for tabs" and "Show line numbers" if you want them, and uncheck "Highlight current line" and "Enable drag and drop":



Under **General⇨Startup and Shutdown** uncheck "Confirm exit when closing last window" and all of the "Plug-ins activated on startup." Also increase the number of recent workspaces under **General⇨Startup and Shutdown⇨Workspaces**.

Under **General⇨Workspace** check "Save automatically before build."

The next step is setting Java-specific options:



Under **Java⇨Code Style** uncheck the "Add '@Override'" box.

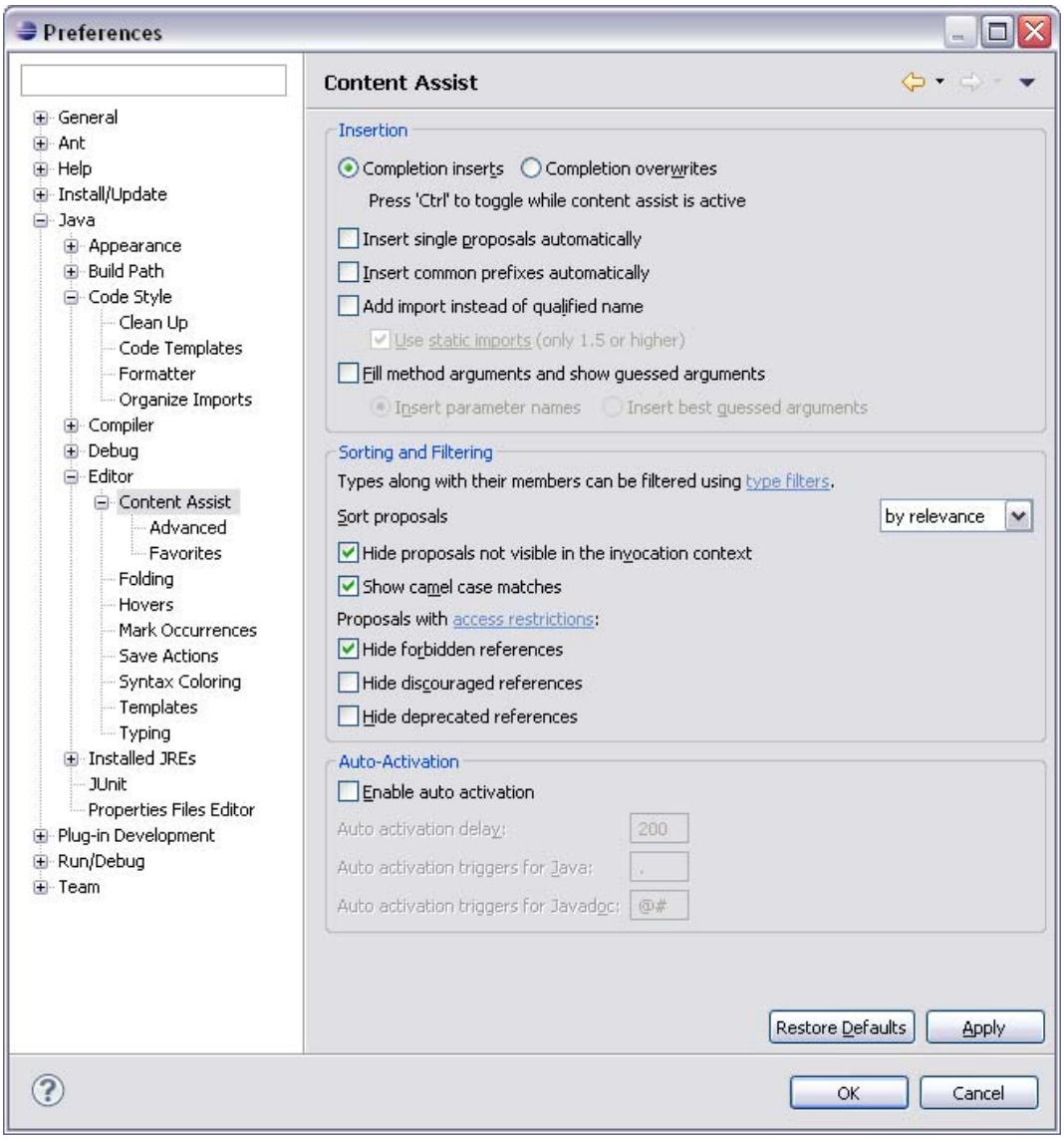
Under **Java⇨Code Style⇨Formatter** click `New`, and enter a name for a new profile:

Click `OK`. Under the "Indentation" tab choose the "Spaces only" tabs policy and set both the indentation size and the tab size to 2. Under the "Braces" tab change all brace positions, except the last one, "Array initializer," to "Next line." Under the "White Space" tab, **Arrays⇨Array initializers** uncheck "after

opening brace" and "before closing brace" boxes. Under the "Control Statements" tab, check all "Insert new line" boxes.

Define another profile under **Java⇨Code Style⇨Clean Up**. Under the "Code Organizing" tab check "Remove trailing whitespace"; under the "Missing Code" tab uncheck "Add missing annotations." Under the "Unnecessary Code" tab uncheck "Remove unused imports."

Under **Java⇨Editor⇨Content Assist** uncheck all the boxes in the Insertion section and uncheck "Enable auto activation":



Under **Java⇨Editor⇨Folding** uncheck all the "Initially fold" boxes or disable folding altogether by unchecking the "Enable folding" box.

Under **Java⇨Editor⇨Mark Occurrences** uncheck the "Mark occurrences" box.

If you do not like italics in your code editor, go to **Java⇨Editor⇨Syntax Coloring**, choose the **Java⇨Static fields** element, and uncheck the "Italic" box.

Under **Java⇨Compiler⇨Errors/Warnings** expand "Potential programming problems" and change "Serializable class without serialVersionUID" from "Warning" to "Ignore."

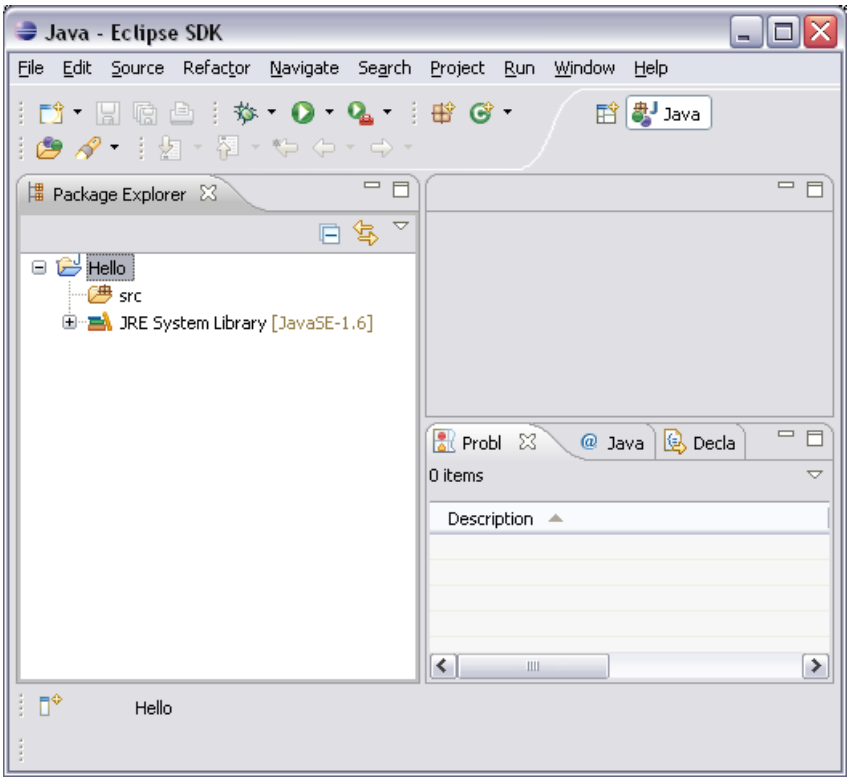
When you are finished setting the preferences, click `OK`. If Eclipse asks you whether it is OK to reload the workspace, click `Yes`.

5. Running "Hello World"

On the `File` menu choose `New⇨Java Project` (or click on the pull-down arrow next to the `New` button on the toolbar and choose `Java Project`). A dialog box pops up. Enter the project name, for example, "Hello"; leave the "Use default location" box checked:

Click `Finish`.

If you expand the `Hello` folder in Package Explorer, you will see the `src` subfolder:



That's where Java source files go. If you are staring from scratch, click on the `New Java Class` button on the toolbar. In the dialog box that pops up, enter the name for your class (for example, `HelloWorld`) and the features you want automatically generated for your class (for example `public static void main`). Click `Finish`. Enter the code for your class in the editor:

```
/**
 *  Displays a "Hello World!" message on the screen
 */

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

From the `Run` menu choose `Run` or press `Ctrl+F11`. This will build or rebuild your project, if necessary, and run it. When run your program for the first time , Eclipse asks you: Run as Java applet or application? Choose "application."

6. Command-Line Arguments and User Input

In the same project "Hello," create two more classes (*Java Methods* Section 2.4), `Greetings` —

```
/**
 *  This program expects two command-line arguments
 *  -- a person's first name and last name.
 *  For example:
 *  C:\Mywork>java Greetings Annabel Lee
 */
public class Greetings
{
  public static void main(String[] args)
  {
    String firstName = args[0];
    String lastName = args[1];
    System.out.println("Hello, " + firstName + " " + lastName);
    System.out.println("Congratulations on your second program!");
  }
}
```

— and `Greetings2`:

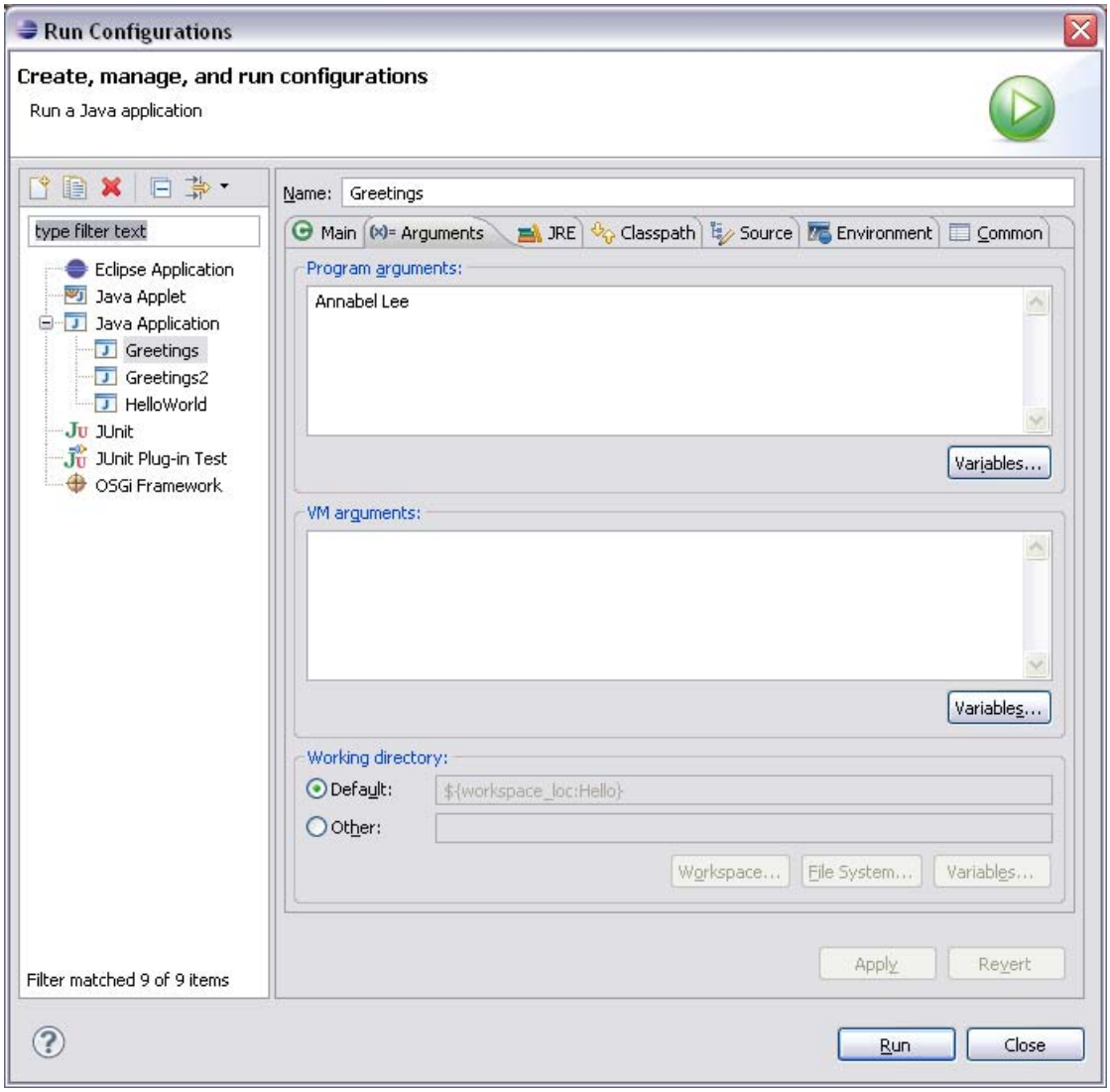
```
 /*
     This program prompts the user to enter his or her
     first name and last name and displays a greeting message.
     Author: Maria Litvin
 */

 import java.util.Scanner;

 public class Greetings2
 {
   public static void main(String[] args)
   {
     Scanner kboard = new Scanner(System.in);
     System.out.print("Enter your first name: ");
     String firstName = kboard.nextLine();
     System.out.print("Enter your last name: ");
     String lastName = kboard.nextLine();
     System.out.println("Hello, " + firstName + " " + lastName);
     System.out.println("Welcome to Java!");
   }
 }
```

`Greetings` expects command line arguments and `Greetings2` accepts input from the user.

If your application expects run-time parameters from the command line, you need to define a run-time configuration. From the `Run` menu choose `Run Configurations` and click on the "(x)=Arguments" tab. Enter the program arguments in the top text area. For example:



Then click `Run`. You have to do it once, as long as you keep the same program arguments.

You can double click on any Java file to open it in the editor window.

> **If you find the Package Explorer view too cluttered or want to see the contents of the `bin` folder (where the `.class` files are placed), from the `Window` menu, choose `Show View`⇨`Navigator`.**

> **It is easy and convenient in Eclipse to have several applets and/or applications in the same project and choose which one of them to run.**

If you press `Ctrl+F11` when a Java class is selected or open it in the editor, Eclipse will run the selected class. If there are several classes that have a `main` method in them and nothing is selected, Eclipse will ask you which one to run.

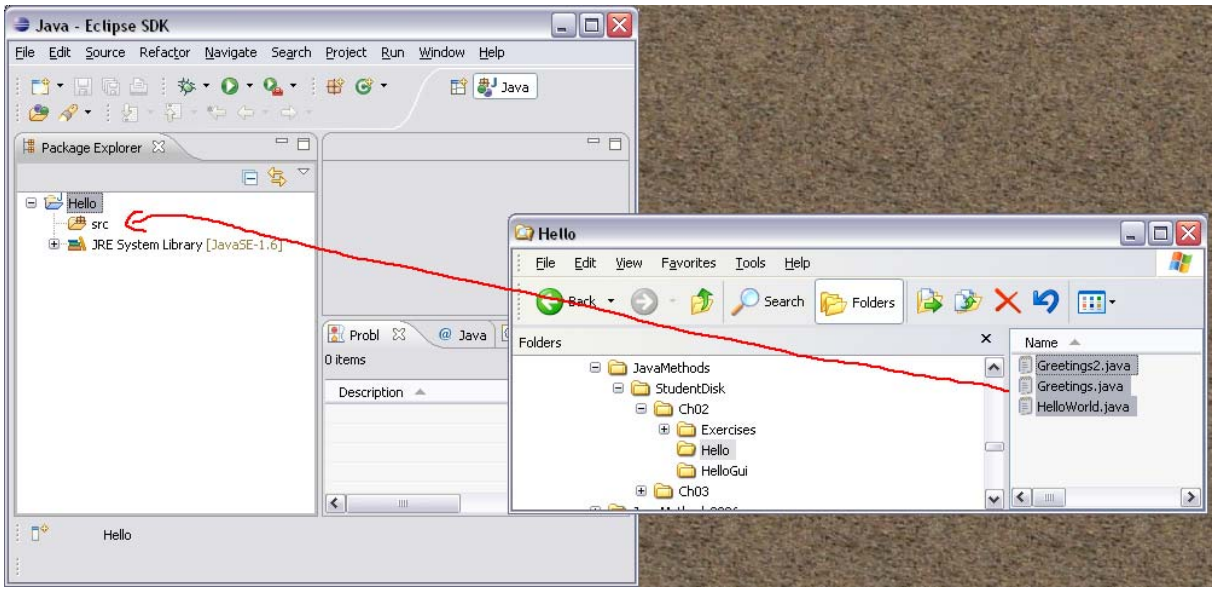
The third application in our "Hello" project, `Greetings2`, prompts the user for input.

Unfortunately, Eclipse doesn't position the cursor correctly on the input line in the console window. If the cursor is in an editor window and you start typing, you will mess up the source code. Make sure to click on the console window before responding to the prompt.

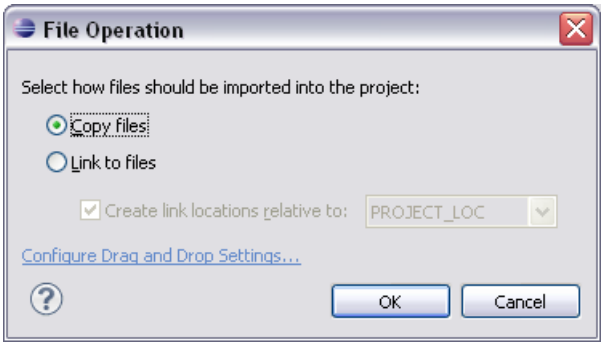
If you program reads data files (`.txt`, `.wav`, etc.) place them into the project folder (such as `Hello`, one step above `src` and `bin`).

7. Bringing Existing Java Files into Eclipse

Suppose `HelloWorld.java`. `Greetings.java`, and `Greetings2.java` already exist on your computer (entered earlier or downloaded with teacjher files. You want to bring them into an Eclipse project. Create the project first. Then open your operating system's file manager (for example, Windows Explorer) outside Eclipse. Select the desired `.java` files and drag and drop (or copy and paste) them into the `src` folder in the project.



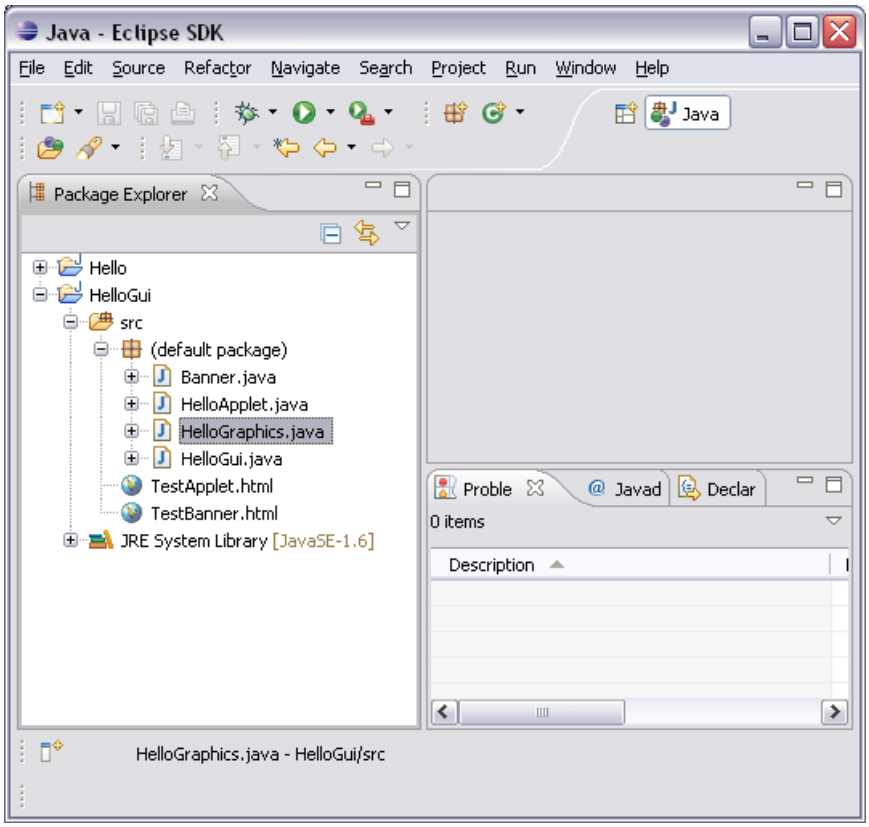
A pop-up dialog will ask you whether you want to copy or "link" the files:



Choose "Copy" to copy the Java files into the project folder. If you chose "Link" you will work with the original files in their original location and you may accidentally ruin or delete them.

8. Running GUI Applications and Applets

Create another project in the same workspace. For example, create a new project named "HelloGui" and copy (drag and drop) all the files from the *Java Methods* Chapter 2 `HelloGui` folder to the project's `src` folder:



Click on any class to select it, then press `Ctrl+F11` to run it.

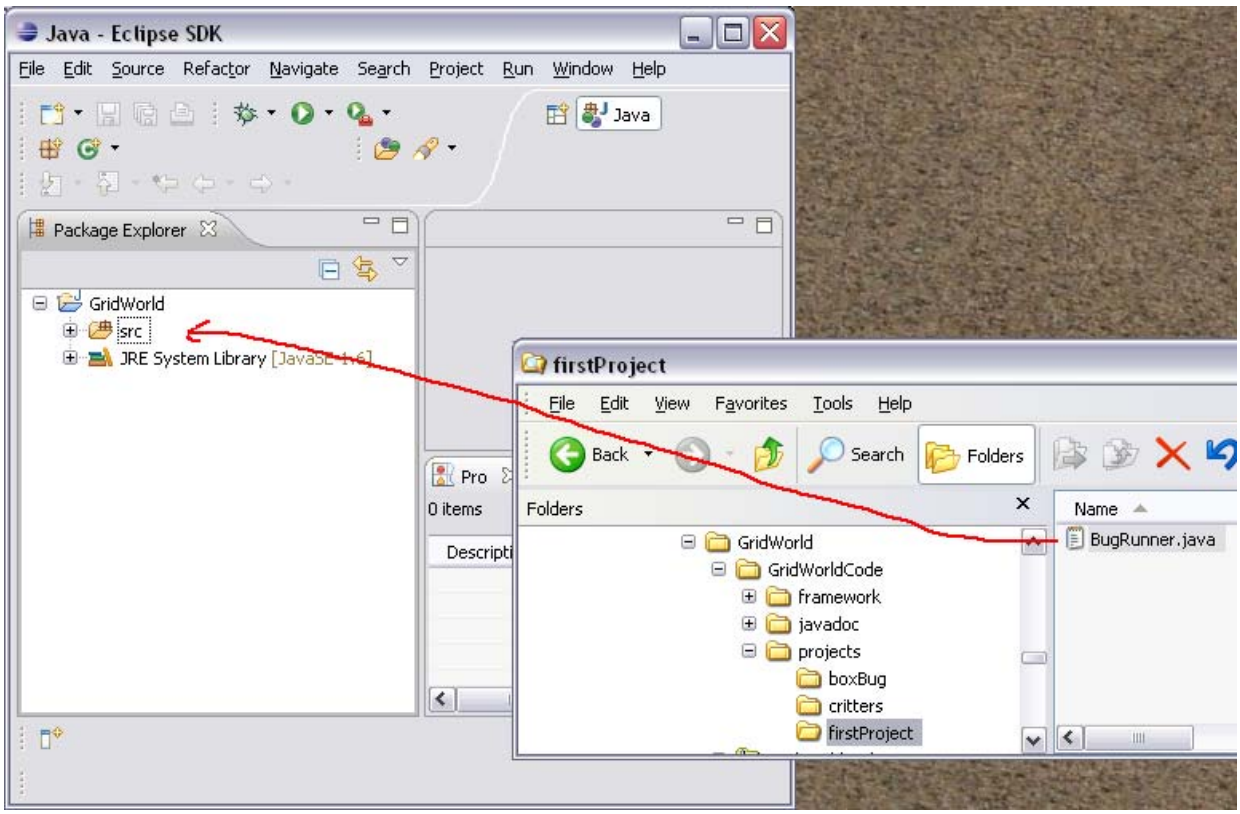
Eclipse does not need an `html` file to run an applet in Applet Viewer. If you supply your own `html` file in the `src` folder, Eclipse will copy it into the `bin` folder when it builds your project. (Eclipse does it to any file that is not Java source.) You can run your own `html` file if you select it (click on it) and press `Ctrl+F11`. But if you double click on an `html` file, Eclipse tries to interpret the `html` tags and, ironically, can't find Java. To examine or edit an `html` file, right click on it and chose **Open With⇨Text editor** or, for a permanent change, choose `Preferences` on the `Window` menu, go to **General⇨Editors⇨File Associations** and add "Text editor" to the `htm` and `html` file types (and click on `Default` to make it the default editor).

(You can test your own `html` files in a browser, of course, outside Eclipse. Just navigate in your file manager to the project's `bin` folder and double click on the `html` file.)

9. Using Jar Files and Running *GridWorld* Programs

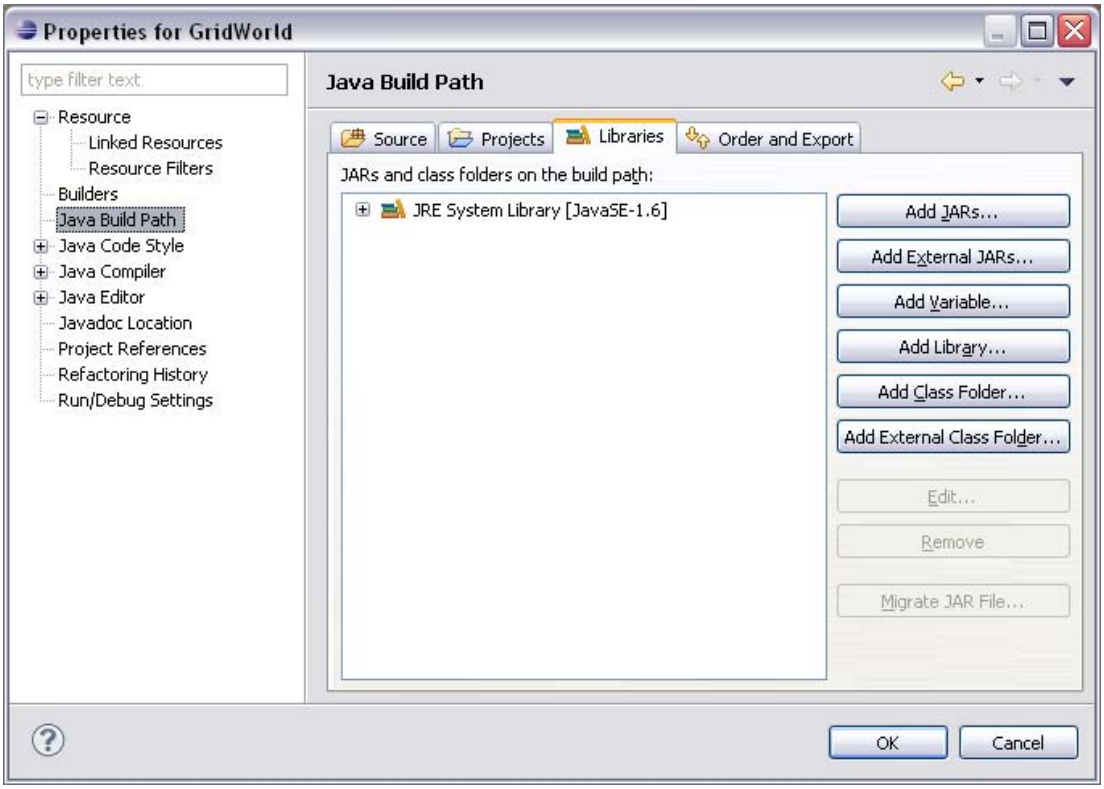
GridWorld case study was developed by the College Board's AP* Computer Science Development Committee for AP CS courses and exams. We will use it as an example for setting up Eclipse projects with third-party `jar` files. *GridWorld*'s materials can be downloaded from the `collegeboard.com`, the AP CS page.

Let us set up a *GridWorld* project in a different workspace: `javamethods/Ch03`. To switch to a different workspace, choose the `Switch Workspace` command on the `File` menu. Import the preferences into the new workspace (see Section 3). Create a new project, called "GridWorld." From the *GridWorld*'s installation folder `GridWorldCode/projects/firstProject` drag and drop the `BugRunner.java` file into the Eclipse project's `src` folder:



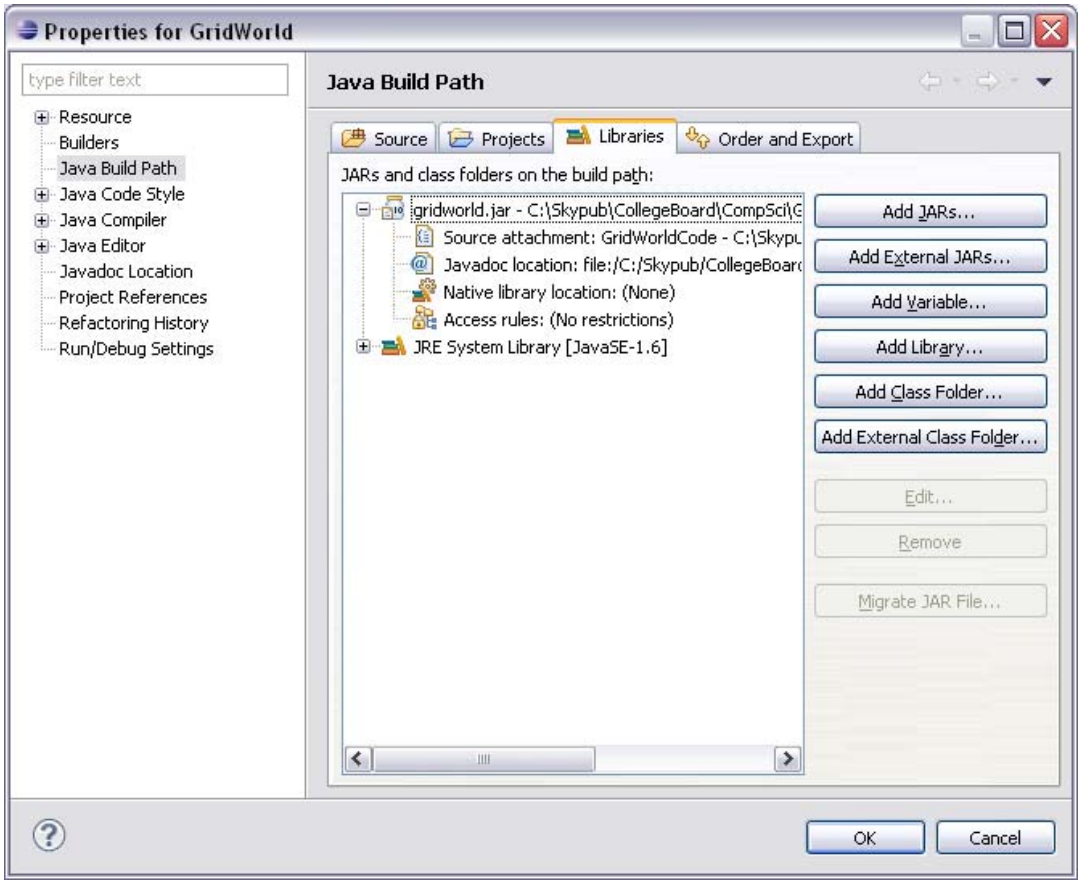
Eclipse will show several errors because *GridWorld*'s library (`gridworld.jar`) is not declared in the project. To add `gridworld.jar`, select the project then choose `Properties` on the `Project` menu (or right-click on the project and choose `Properties` or select the project and press `Alt-Enter`). Select **Java Build Path** and click on the "Libraries" tab:

* AP and the Advanced Placement Program are registered trademarks of the College Entrance Examination Board; their use does not constitute endorsement of this document by the College Board.



Click `Add External JARs`, navigate to the *GridWorld* installation folder `GridWorldCode` and choose `gridworld.jar`.

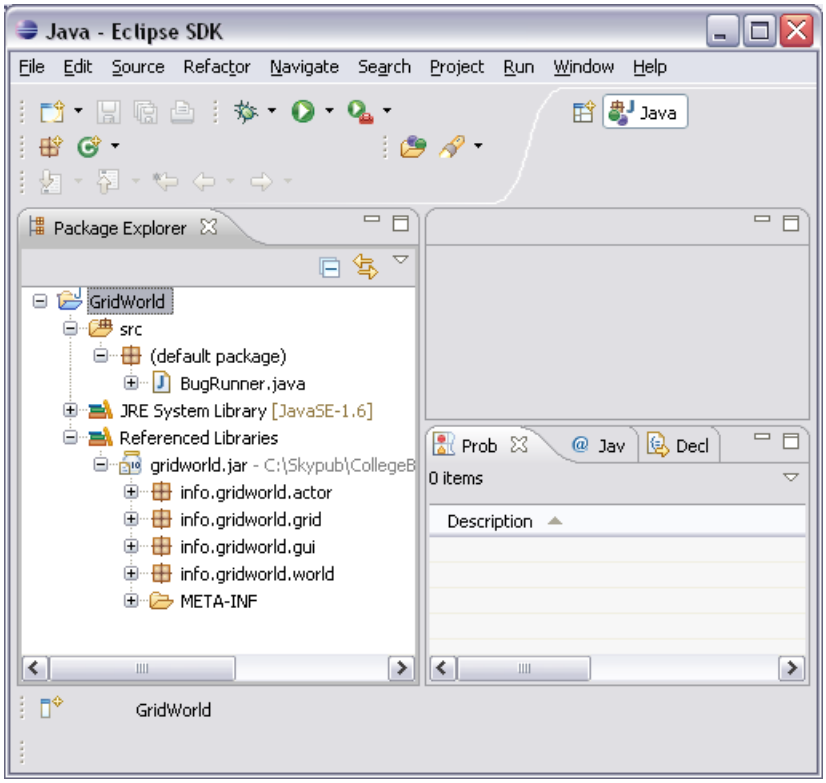
This would be sufficient for running a *GridWorld* project, but to take full advantage of Eclipse's interactive tips and to have convenient access to *GridWorld*'s source code, it is necessary to "attach" the source code and javadoc to the jar. In the same dialog, expand the newly created "gridworld.jar" line —



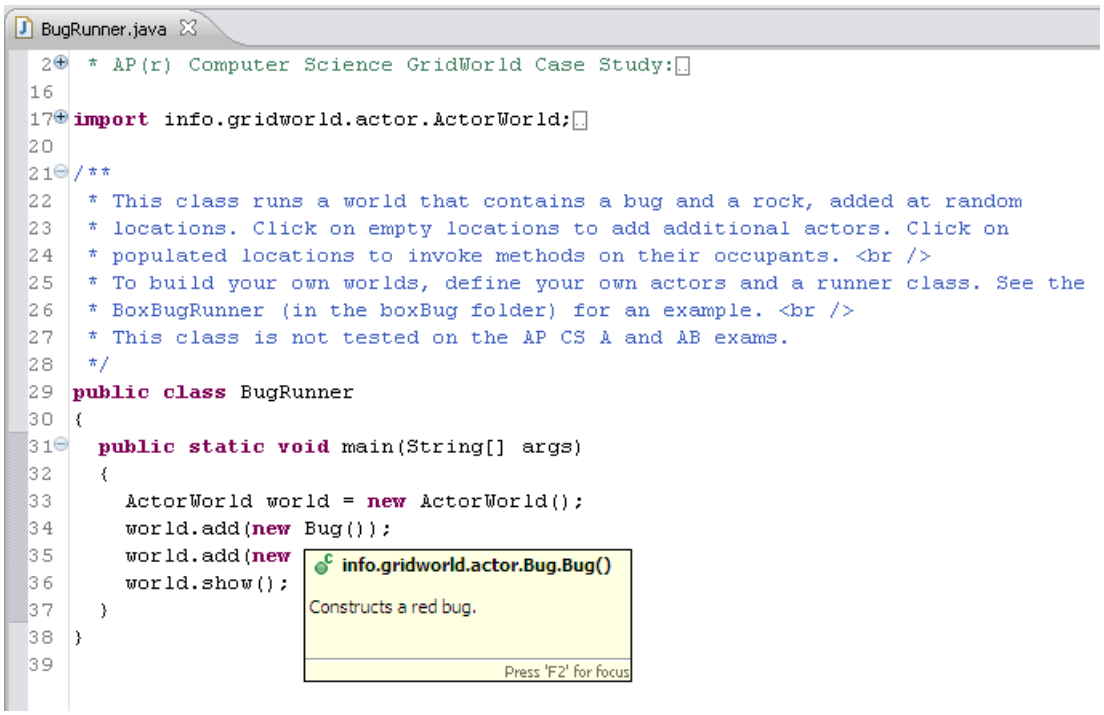
— click on "Source attachment," and click `Edit...` You will see a source attachment pop-up:

Navigate to the external folder that contains *GridWorld* source (`GridWorldCode`) and click `OK`. Repeat the same for "Javadoc location": navigate to `GridWorldCode/javadoc` and click `OK`. Click `OK` when you have added `gridworld.jar` and attached the source and javadoc to it.

Now the errors disappear, you can see a "Referenced Libraries" entry in the Package Explorer, and you have access to the *GridWorld*'s classes and the source code:



If you hover over a *GridWorld* element in the editor, Eclipse displays a tip from *GridWorld*'s javadoc:



(Like almost everything else, "hovers" are configurable in Eclipse.) Pressing `F2` redisplays the javadoc tip. Pressing `Shift+F2` opens the full javadoc window in the editor.

Press `Ctrl+F11` to run the project.

10. Content Assist and the Debugger

Eclipse offers many features that speed up typing and correcting the mistakes. The Content Assist feature provides context-sensitive code completion. Press `Ctrl+Space` to see the code completion suggestions (called *proposals*). For example, if you type `syso` and press `Ctrl+Space`, Eclipse will suggest `System.out.println();`. We think Content Assist may confuse a novice; a student has to acquire some familiarity with Java to employ this feature effectively.

As far as the debugger is concerned, see Top Ten Reasons Not to Use a Java Debugger in School.