How to Set Up Java Swarm in Eclipse

Prepared by: Steve Lytinen (http://condor.depaul.edu/~slytinen/) and Steve Railsback (Lang, Railsback & Associates)

June, 2006

Disclaimer: This document is provided in hopes that it will be useful to agent-based modelers, but with no guarantee that it is accurate, up to date, or even helpful. We recommend you start by trying to follow our instructions exactly; if they fail, then experiment or get help. If you find mistakes or improvements, please contact us through the Swarm Development Group wiki (www.swarm.org) page where this document is posted, or post your comments to the agent-based modeling email list (see: www.swarm.org/wiki/Swarm:_Mailing_lists).

Updated May 2007 to change back- to front-slashes in SWARMHOME and PATH.

I. Introduction

Eclipse is a widely used and powerful integrated development environment, and makes a convenient platform for developing Java Swarm models. Eclipse is free and available for Linux and Windows. Eclipse provides features such as a Java-specific editor, graphical displays of program and package hierarchies, and a debugger. This how-to describes setting up Eclipse so the Java Swarm libraries can be viewed and accessed in a new model. It was written for version 2.2 of Java Swarm and Eclipse 3.1 for Windows.

Notes:

- The approach is to set up JavaSwarm as an Eclipse project; then set up new projects for your Java Swarm models. Your new projects will access the JavaSwarm project.

- If you deviate from any of these instructions, your setup is very likely to fail.

- When you import code (including Swarm, or an existing Swarm model) into Eclipse, it creates a complete new copy of the code in your Eclipse workspace. This means you will end up with two copies of Swarm, which also means that you can delete models or Swarm from Eclipse without necessarily removing them from your computer.

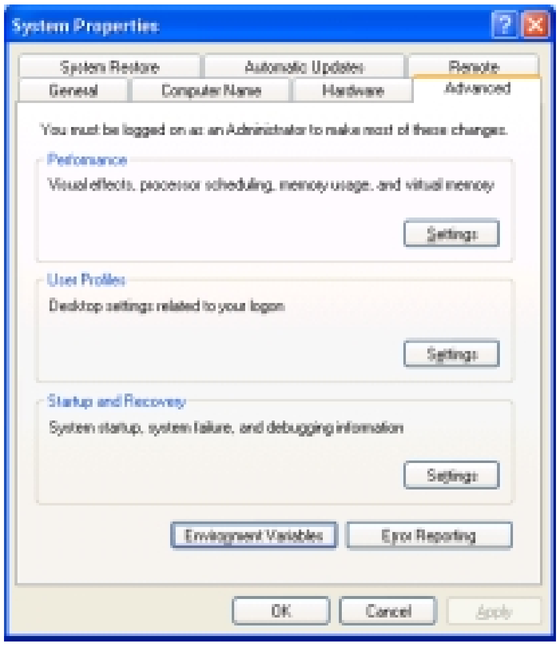
- If you need to start over, be sure to first delete the folders for Java Swarm and any other projects you created in the Eclipse workspace directory, by right-clicking on them in the Package Explorer panel of Eclipse. Tell Eclipse to delete the project's contents too.

II. Installing Eclipse and Java Swarm in Eclipse

1. Install Eclipse, using downloads from: http://www.eclipse.org/. Install JavaSwarm, by downloading http://ftp.swarm.org/pub/swarm/binaries/w32/Swarm-2.2-java.zip and unzipping it into c:\Swarm-2.2-java

2. JavaSwarm requires two Windows environment variables: `SWARMHOME` and `PATH.` Almost all Windows machines will already have a `PATH` environment variable; the `SWARMHOME` environment variable will be new (unless you have already been using Swarm).

    a. Go to the Windows Control Panel, and select "System". Click the "Advanced" Tab, and then "Environment Variables".



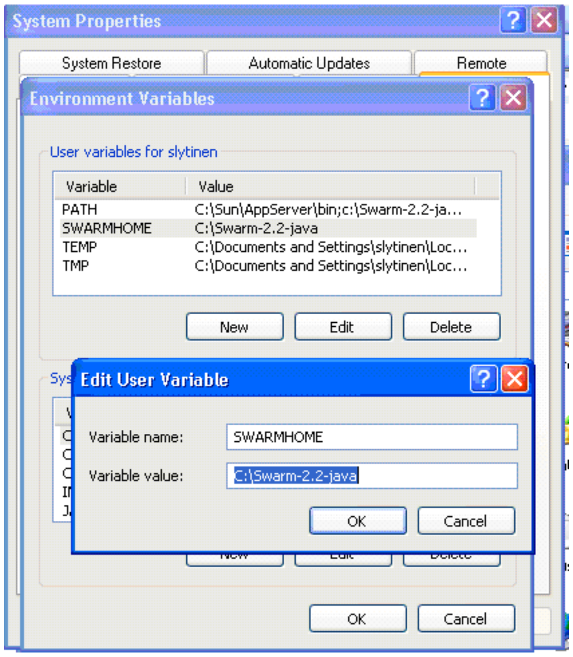
You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your logon

System startup, system failure, and debugging information

    b. To create the `SWARMHOME` environment variable, click on "New", either in the "User variables" or "System variables" portion of the "Environment Variables" window. In the panel which pops up, fill in the Variable name as `SWARMHOME` and the Variable value as `c:/Swarm-2.2-java` (or wherever you installed Java Swarm). Then click "OK" in the "New Variable" window. (NOTE: Despite the figure below, which is wrong, use a forward slash, not a backslash.)

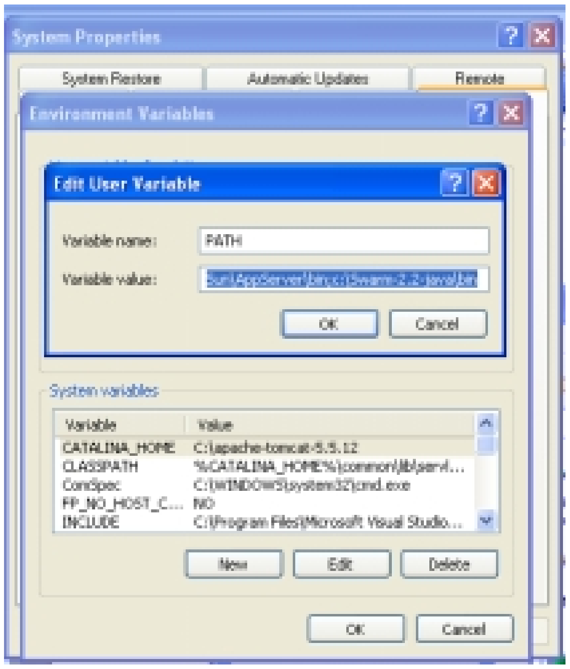


User variables for slytinen

Variable	Value
PATH	C:\Sun\AppServer\bin;c:\Swarm-2.2-ja...
SWARMHOME	C:\Swarm-2.2-java
TEMP	C:\Documents and Settings\slytinen\Loc...
TMP	C:\Documents and Settings\slytinen\Loc...

Variable name: SWARMHOME
Variable value: C:\Swarm-2.2-java

c. To edit the PATH environment variable, highlight the PATH variable, either in the "User variables" or "System variables" portion of the "Environment Variables" window, and click "Edit". In the panel which pops up, edit the Variable value to include `c:/Swarm-2.2-java/bin` (or wherever you installed Java Swarm). Remember that there must be a semicolon between each portion of PATH. (NOTE: Despite the figure below, which is wrong, use forward slashes, not backslashes, for this addition to your path.)



Variable name: PATH
Variable value: Sun\AppServer\bin;c:\Swarm-2.2-java\bin

System variables

Variable	Value
CATALINA_HOME	C:\apache-tomcat-5.5.12
CLASSPATH	%CATALINA_HOME%\common\lib\servl...
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
INCLUDE	C:\Program Files\Microsoft Visual Studio...

d. Finally, click "OK" in the "Environment Varibles" window.

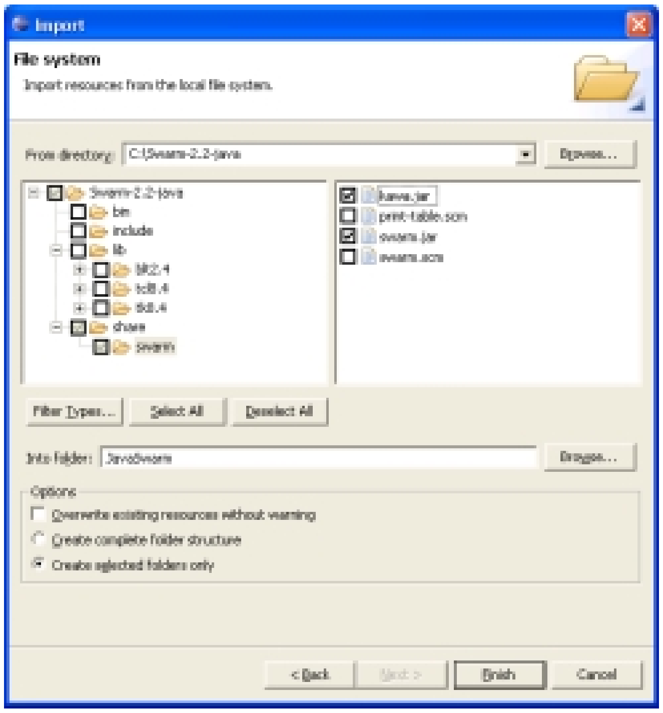
3. Start Eclipse. First, Eclipse will ask you to select a directory for its workspace: the place where it stores the program files you create. Then, click on "Workbench" (an icon in the upper right corner of the Welcome screen) to go to the Eclipse workbench.

4. Create a Java project for JavaSwarm.

   a. In the Eclipse menu, select `File -> New -> Project`. This opens a window where you select the `Java Project` wizard and click `Next`.

   b. In the new project window that opens, provide the project name (assumed here to be "JavaSwarm"), and click on `Create new project in workspace`, and `Create separate source and output folders`. Then click on `Finish` to close the window.

   c. The new JavaSwarm project now appears in Eclipse's Package Explorer window. Right click on the JavaSwarm project and select `Import -> File system -> Next`. Browse to the Swarm-2.2-java directory on your computer. (With a typical Windows installation, this will be C:\Swarm-2.2-java.) Click `OK`, then click the + to expand the "Swarm-2.2-java" and "share" folder icons. Then click on the "swarm" folder icon to view its contents. Check the boxes for `kawa.jar` and `swarm.jar` on the right side of the Import window, as shown below. Then click "Finish".



From directory: C:\Swarm-2.2-java

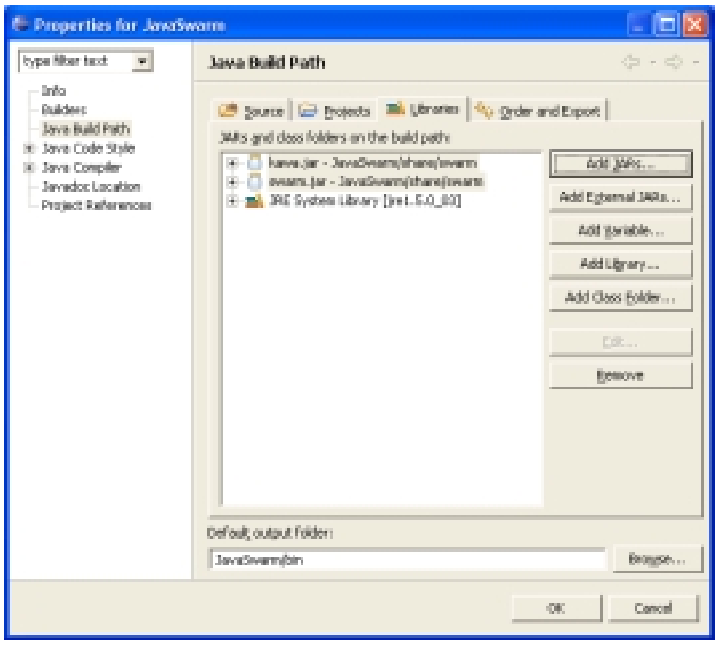
- ☐ Swarm-2.2-java
  - ☐ bin
  - ☐ include
  - ☐ lib
    - ☐ blt2.4
    - ☐ tcl8.4
    - ☐ tk8.4
  - ☐ share
    - ☐ swarm

- ☑ kawa.jar
- ☐ print-table.scm
- ☑ swarm.jar
- ☐ swarm.scm

Into folder: JavaSwarm

Options
- ☐ Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

d. In the Package Explorer window, right-click on the JavaSwarm project. Select `Properties -> Java build path` and click on the `Libraries` tab. Click `Add JARS`. Expand the “JavaSwarm”, “share”, and “swarm” folders, and select kawa.jar and swarm.jar. Leave the “Default output folder” as it is. Click `OK`. When you are done, the “Properties for JavaSwarm” window should look like this:



JARs and class folders on the build path:

kawa.jar - JavaSwarm/share/swarm
swarm.jar - JavaSwarm/share/swarm
JRE System Library [jre1.5.0_03]

Default output folder:
JavaSwarm/bin

e. Still in the `Properties -> Java Build Path` window, select the `Order and Export` tab. Click `Select All`, then de-select the JRE system library. Click `OK`.

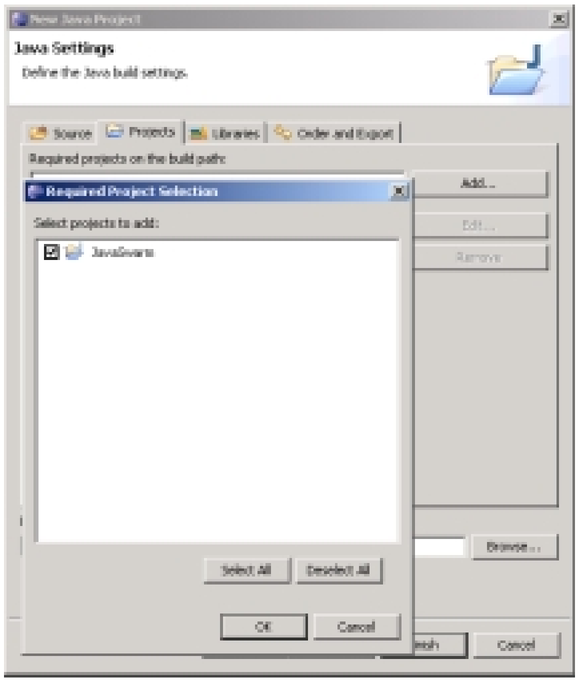
III. Building New and Running Existing Swarm Models

5. Build a new project for your Java Swarm model. Use this step if you want to create a new Java Swarm model from the start. (Use step 6 to import and modify an existing Java Swarm model.)

   a. In the Eclipse menu, select `File -> New -> Project`. Then, in the "Select a wizard" window, select `Java Project` and hit `Next`.

   b. In the new project window that opens, provide a name for your project, and click on `Create separate source and output folders`. Click `Next` to move to the `Java settings` window.

   c. Click on the `Projects` tab, and hit `Add`. Click on the JavaSwarm check box to add it to your new project. Click on `OK`, then `Finish` to close the window. This makes all the Swarm classes accessible to your new project.



d. Create a new Java package for your model. In the Eclipse Package Explorer window, expand the new project you just created and click on its “src” folder. (If there is no “src” folder, you forgot to hit `Create separate source and output folders` in step 5 b, above.) Then on the Eclipse menu, hit `File -> New -> New Java Package` (not "project"). Enter a valid package name (Eclipse will tell you if you try to use an invalid name, such as one starting with a capital letter).

   Now, you can start using Eclipse tools to create new classes, etc. By expanding the JavaSwarm project in the Package Explorer window (expand “swarm.jar” after expanding the JavaSwarm project), you can see all the JavaSwarm classes and methods that you can use.

6. Import existing code, including an Eclipse project. If you want to import an existing Java Swarm model (including one developed in Eclipse), follow these steps.

   a. In the Eclipse menu, select `File -> New -> Project`. Then, in the "Select a wizard" window, select `Java Project` and hit `Next`.

   b. In the new project window that opens, provide a name for your project, and click on `Create separate source and output folders`. Click `Next` to move to the `Java settings` window. (Do *not* click `Create project from existing source`—it will try to create the Eclipse project in whatever directory your source code is already in.)

c. Click on the `Projects` tab, and hit `Add`. Click on the JavaSwarm check box to add it to your new project. Click on `OK`, then `Finish` to close the window.

d. See (f) below if you want to import an existing Eclipse project. If instead you are importing a directory of existing .java files, first create a new package within the new project. In the Eclipse Package Explorer window, expand the new project you just created and make sure it contains a “src” folder. (If there is no “src” folder, you forgot to hit `Create separate source and output folders` in step 6 b, above.) Then on the Eclipse menu, hit `File -> New -> New Java Package` (not "project"). Enter a valid package name (Eclipse will tell you if you try to use an invalid name, such as one starting with a capital letter). This will create a new package directory under “src”.

e. In the Eclipse Package Explorer, right-click on the new package and select `Import -> File system`. In the window that opens, browse to and select the directory of source code to import. After you select the directory, Eclipse will let you select some or all of the files in the directory.

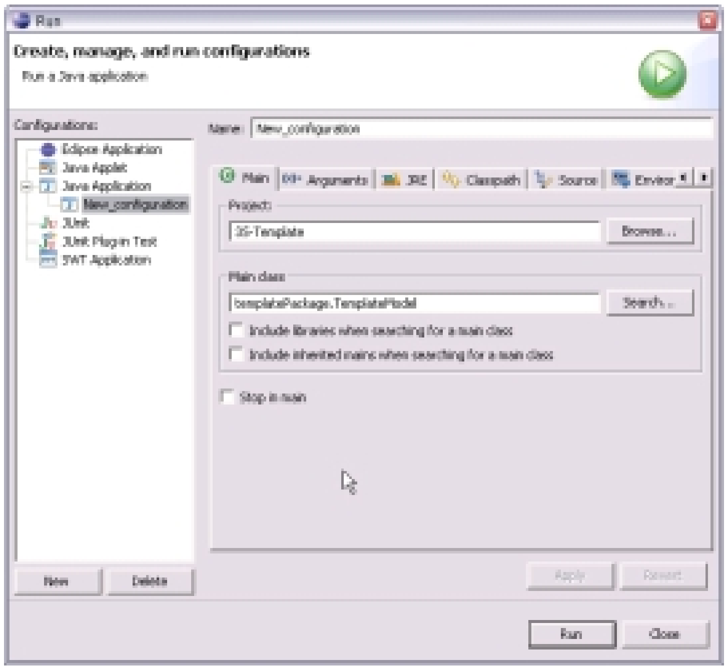
   One common problem at this point is that the existing source code you import has no package statements, or wrong package statements: at the top of each .java file must be a statement “`package thePackageName;`” where `thePackageName` must match the package name in the Eclipse Package Explorer panel.

f. If you are importing an Eclipse project, just select the project's entire directory. A checkbox appears for the project; click it to get the whole project. Click "no" for whether to overwrite the existing class path, but "yes" to overwrite other stuff.

IV. Running Models

7. To attempt compiling and running a model in Eclipse:

   a. On the Eclipse main menu, hit `Run -> Run`. This opens a window in which you select a "run configuration"- a set of code you want to execute. First, on the left side of the window select `Java Application`. Then hit `New`, and enter the name of the package and the name of the file containing its Main class. Then hit either `Apply` or `Run`.


Run

Create, manage, and run configurations

Run a Java application

Configurations:

Name: New_configuration

Project: JS-Template

Main class: templatePackage.TemplateModel

Include libraries when searching for a main class

Include inherited mains when searching for a main class

Stop in main

Your application will now attempt to compile and run, with errors appearing in the Problems window.

b. You can re-run a program, once configured, by hitting `Run -> Run Last Launched` on the Eclipse menu.

V. Using the Eclipse Debugger

NOTE: Remember that Swarm's Java classes are really just wrappers around Swarm's library code, which is in Objective-C. This means that you cannot see the Swarm source code in Eclipse and you cannot use the debugger to see what is going on when your model is executing Swarm library code—you can only see the Java code you wrote.

1. Set a breakpoint: a place in your code where the debugger will stop and let you inspect the model's state. In the Eclipse code editor, right-click in the left margin at the line where you want a breakpoint and select `Toggle Breakpoint`. A red message will appear at the bottom of the Eclipse window if it is not a valid place to put a breakpoint. (If right-clicking does not work, click to put the cursor on the line where you want a breakpoint; then use the menu to hit `Run -> Toggle Line Breakpoint`.)

2. On the menu, select `Run -> Debug`. A window appears to select which run configuration to debug; the default is the last thing you tried to run. Hit `Debug`. Your Swarm model will start up; click the Swarm start button to begin execution. The code will run until it hits a breakpoint.

3. A pop-up window will ask you to confirm that it is OK to switch to the "debug perspective" (a different arrangement of Eclipse windows for debugging). It is.

4. A "Debug" window will appear. It has buttons on its toolbar that let you "step into" (execute one line at a time, going into the method calls as they are encountered), "step over" (skip over method calls), and "Step Return" (which runs until the next Return statement is reached - use this when the debugger delves into a bunch of very strange, confusing Java thread stuff that you do not want to know about). See the Eclipse documentation for more details on stepping the debugger.

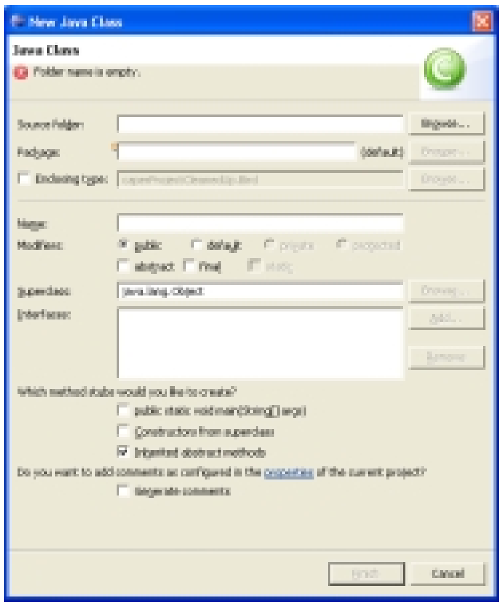
5. In the Variables window, you can expand objects to see values of their instance variables.

6. To terminate the debugger, click on the red square button in the Eclipse Debug window. Then, you need to right-click on anything in the Debug window and hit Remove all terminated to get rid of old debug threads.

VI. Eclipse Features to Help You Write and Debug Java Code

There are several reasons to use an Integrated Development Environment (IDE) such as Eclipse when writing code. Here, we discuss some of the features provided by Eclipse that make writing Java programs easier.

1. Declaring a new class
    a. Click on the "New Java class" icon. You will then see a window that looks like this:



New Java Class

Java Class

Folder name is empty.

Source Folder:
Package: (default)
☐ Enclosing type:

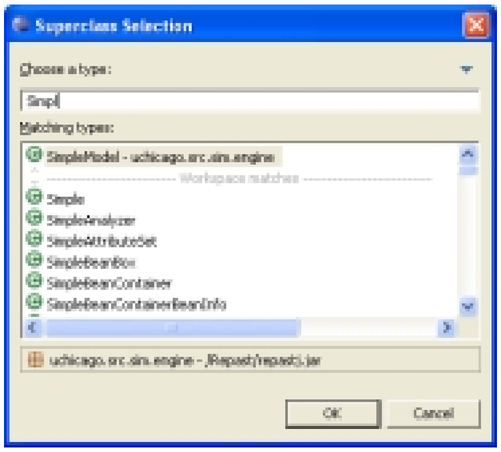
Name:
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments as configured in the properties of the current project?
☐ Generate comments

Finish Cancel

Probably, the "Source folder" and "Package" textboxes will automatically be filled in.

    b. Type a class name (in the "Name" textbox). If you wish to extend a class other than Object, click the "Browse" button (it will become clickable once you type a class name). Then you will see a new Window. As you start typing in the name of the desired superclass, you will see a list of built-in (Swarm or Java) classes that match what you have typed. Select the class you want, and click OK.



Simpl

SimpleModel - uchicago.src.sim.engine
------------ Workspace matches ------------
Simple
SimpleAnalyzer
SimpleAttributeSet
SimpleBeanBox
SimpleBeanContainer
SimpleBeanContainerBeanInfo

uchicago.src.sim.engine - /Repast/repast.jar

c. If you want your class to implement an interface, click the “Add” button next to the Interface text area. Filling in an interface name is similar to selecting a superclass.
d. Click the appropriate checkboxes (for example, if you want your class to have a main method); then click the “Finish” button.

2. Adding import statements

   If you write code that uses a class in another package (including the Swarm packages), it is necessary to have an import statement at the top of the source file. Eclipse will automatically generate the appropriate import statement if you:
   a. Highlight the class name. You can do this by double-clicking with the cursor over any part of the name.
   b. In the Source menu, select “Add import”.
   c. Select the appropriate package/class combination to import.

3. Generating “setter” and “getter” methods

   There are many situations in which methods are written to set or return the value of an instance variable. These methods are called “setter” and “getter” methods.

   a. Highlight the variable name, as described above.
   b. In the Source menu, select “Generate getters and setters”.
   c. Click/unclick the checkboxes that are displayed, depending on whether you want both a getter and a setter, or just one or the other.

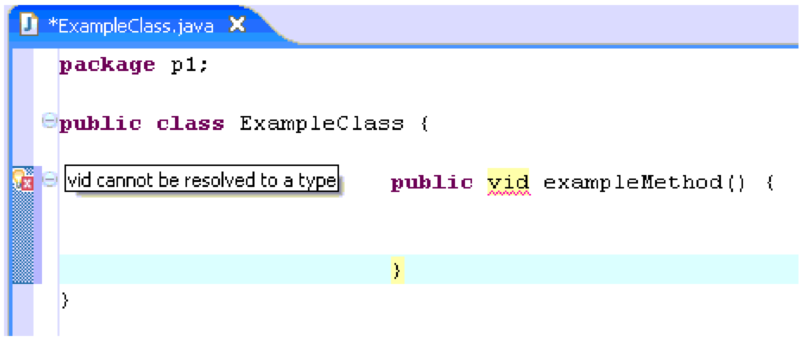
4. Renaming

   If you ever decide that you want to rename something in a Java program, it is easy to do in Eclipse. Simply highlight the item (variable, class, package, etc.) that you wish to rename.

Then, in the Refactor menu, choose “Rename”. Eclipse will propagate the name change throughout the entire project.

5. Correcting syntax errors

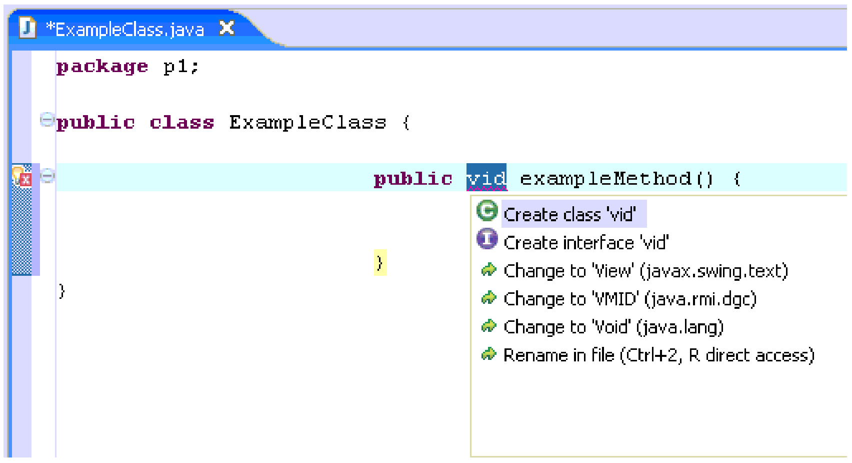
If you miswrite a Java statement, Eclipse will mark the line with a red X symbol on the left edge of the code window. To see what the mistake is, move the cursor over the X. A message will appear that tells you what is wrong. For example:



```
package p1;

public class ExampleClass {

        public vid exampleMethod() {

        }
}
```

vid cannot be resolved to a type

Eclipse can sometimes propose a fix. To see possible fixes, click on the red X:


*ExampleClass.java
Create class 'vid'
Create interface 'vid'
Change to 'View' (javax.swing.text)
Change to 'VMID' (java.rmi.dgc)
Change to 'Void' (java.lang)
Rename in file (Ctrl+2, R direct access)

```
package p1;

public class ExampleClass {

        public vid exampleMethod() {

        }
}
```

Select the one which will fix the problem (in this case, "Void").