Chapter 6:
Running and Debugging an Applet

Chapter introduction

A compiler will catch most errors your code may have in syntax; but, because the computer doesn't already know what you have in mind, it is impossible to detect flaws in the logic of your code. This chapter discusses what to do after you've compiled your code: how to run and debug it.

Running an applet

Once your applet has been successfully compiled, you can run it using the **Roaster Applet Runner** or debug it with the help of the built-in debugger. The Roaster Applet Runner allows you to run your applet without the need of a Java-capable Web browser.

Running applets referenced within HTML files

The Roaster Applet Runner reads an HTML file and displays only the applets found inside of the HTML source (these are indicated by the <APPLET> HTML tag). If there is more than one applet reference inside of the HTML file, the Applet Runner will run each applet in a separate window.

Running an applet from the IDE

You can run an applet in a few different ways. The first is to run it from within the IDE. You can accomplish this by selecting **Run** from the **Project** menu or by hitting "CMD-R." This will

cause the IDE to launch the Roaster Applet Runner, which will open the HTML file called “example1.html” in the current project’s folder.

Remember that this file must have an applet tag that references the class of your project. See the “Preparing the Roaster Applet Runner to run your applet” portion of Chapter 3: Organizing and Managing Your Projects for more information about this HTML file.

If the IDE determines that any file associated with your Java project has been modified since you last compiled, it will compile your project again when you attempt to run it. You can prevent this by turning off the **Make all before run** option found in the Compiler panel of the Preferences window. Remember that if you turn off this feature, you must manually recompile your applet by choosing **Compile** or **Make** from the **Project** menu before you try to run it if you’ve made changes to any of the source code.

Running an applet by dragging an icon

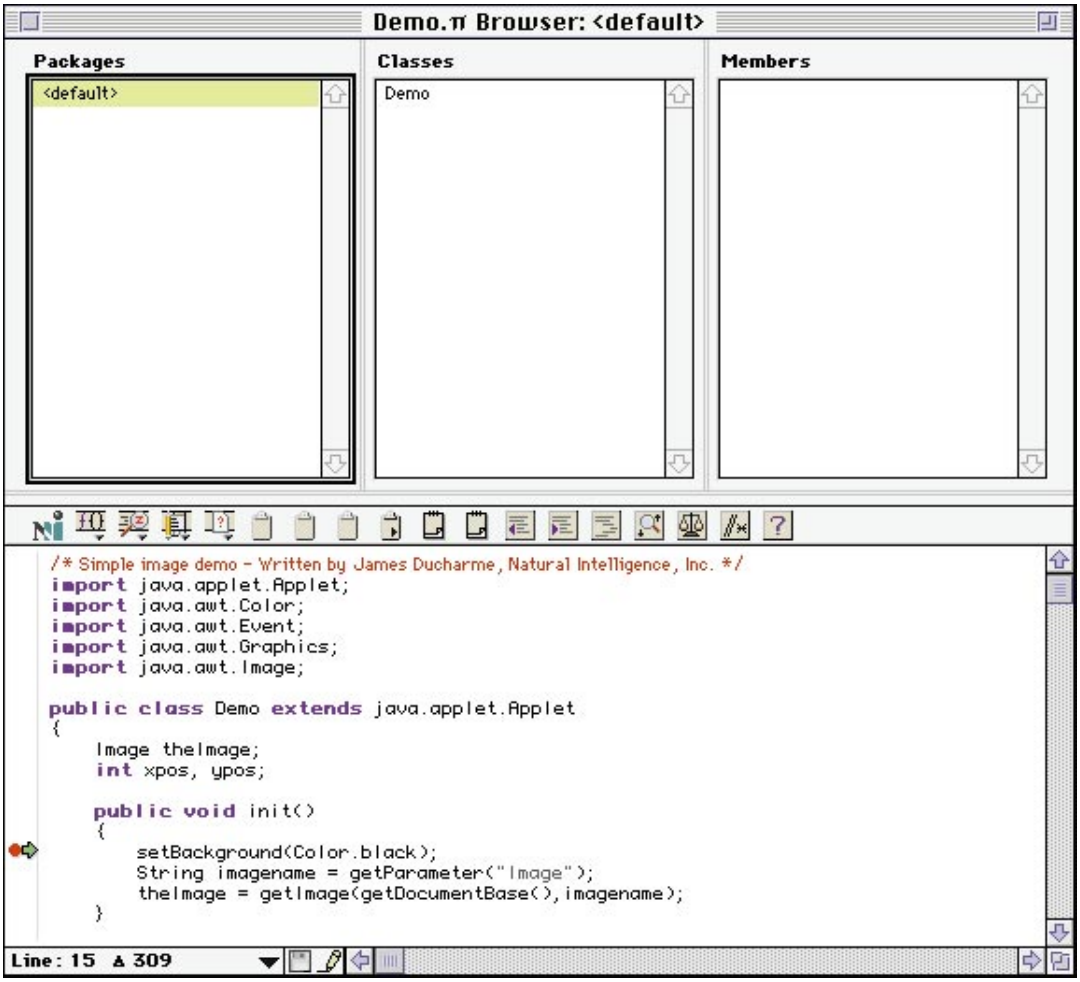
You can also run an applet by dragging either the HTML source file or the project icon onto the Roaster Applet Runner icon. If you run the applet by dragging the HTML source onto the runner, you should first make sure that the .class files needed to run the applet are in the same folder as the HTML source. If you run the applet by dragging the project file onto the runner, make sure that both the .class files and the “example1.html” HTML source file are in the same folder as the project file.

Debugging your code

Now that you know the various ways to run an applet, you can move on to debugging your code. Roaster’s debugger supports both byte code and source code level debugging. (After starting the program and enabling the debugger, simply

control-click and hold in the browser to bring up a popup menu to switch between byte code debugging and source code debugging. Also, you can access this option from the **Debug** menu.) The debugger will allow you to step through your code line by line, set breakpoints at various places in your applet, or view the current contents of objects or variables in the current applet, as well as the current execution stack.

To use the debugger, you must be in the Roaster IDE with the project you want to debug open. You can enable or disable the debugger by selecting **Enable Debugger** from the Project menu. The menu item will appear with a check mark when the debugger is enabled.



With the debugger enabled, you can place breakpoints in your code by clicking in the left hand margin. You can set breakpoints to control where the debugger should halt execution of the applet. To set a breakpoint, you must have the debugger enabled and a .java or .class file open. When you have determined where in the file you want to set a breakpoint, click once to the left of the light gray line in the file. A red dot will appear where you clicked, indicating a permanent breakpoint, which must be removed by the user to become disabled. To set a “one-shot” breakpoint, which will be removed automatically once the debugger reaches that point, simply option-click to the left of the light gray line, and a blue dot will appear to indicate a one-shot breakpoint. With either type of breakpoint, the debugger will stop executing on the breakpoint it encounters and return control to you.

Please note that when the Applet Runner is not running, you can set breakpoints literally anywhere in your files. Once you launch the Applet Runner, it verifies the breakpoints and disables the ones that are invalid. However, when the Applet Runner is running, you can only set valid breakpoints. If you try to set invalid breakpoints (i.e., breakpoints next to a non-excecutable line of code such as comments or simple variable definitions) , you will not be able to.

As mentioned above, permanent breakpoints (set by clicking) need to be removed by the user, and one-shot breakpoints (set by option-clicking) are removed automatically once the debugger reaches that point. If you open the **Show Breakpoints** window, it has the same red (permanent) or blue (one shot) dots on the left side. Clicking these dots toggles between permanent, one shot and disabled breakpoints. A disabled breakpoint is just that, disabled, but not removed. You can turn it on again by clicking the dot area and making it a one shot or permanent breakpoint again.

To remove a breakpoint, simply click on the red dot in your source code window and it will go away. Once your break-

points are set, you can execute your applet by selecting **Run** in the **Project** menu. You can set breakpoints at any time.

The green arrow to the left of the code shows the debugger's current point of execution. The code that the green arrow points to is code that has not yet been executed. At this point, you are ready to step through and debug your code using the debug tools found in the **Debugger Controls** window (see below) or the **Debug** menu.

The debug tools are:

Run: This tells the runtime to continue execution of the applet up until the next breakpoint or until the program completes if no breakpoints are set.

Stop: This will stop execution of the applet if it is running.

Kill: Kills the current applet and quits the Roaster Applet Runner.

Step Over: Executes the current line of code and stops the debugger at the next line. If the line contains calls to a method, that method will be called before returning to the next line in the current method.

Step In: If the current line of code is a call to a subroutine, then the debugger steps into the first line of that subroutine and stops. Otherwise, it has the effect of Step Over.

Step Out: The debugger continues execution until the current subroutine returns to its caller. A typical use of this function is when you step into a function called from your class and you have seen all you need to see in the subroutine, you will want to continue execution of the subroutine until you get back to the previous routine.

Step Over Continuous: This is the same as if you hit the Step Over button repeatedly. This function lets you view the execution of the current class' byte code. Hit Stop or CMD-"." to

halt execution.

Step In Continuous is similar to Step Over Continuous, but instead it steps into every line of code instead of stepping over them. This feature allows you to see the execution of every line of code in your applet.

You can test the debugger using the Demo sample code listed earlier by performing the following steps:

1) Ensure that the Demo project file is open in the IDE and is the current project.

2) Enable the debugger by selecting this item under the Project menu.

3) Open the Demo.java file located in your project directory.

4) Set a breakpoint in the init method on the "setBackground(Color.black);" line and another in the mouseMove method on the "xpos = x;" line by clicking to the left of the code. Make sure a red dot appears next to each line you selected.

Remember that currently, the IDE will allow you to set breakpoints on any line, even lines that are not executed (i.e., comments.) ,Make sure you only set breakpoints on lines of code that are executed, otherwise the debugger will not stop. However, if the Applet Runner is running, you will only be able to set breakpoints on valid lines.

5) Select **Run** from the **Project** menu. The Roaster Applet Runner will start up and a browser window will open.

When you run an applet, the AppletViewer.class file will appear in the browser because this is where Roaster begins execution of an applet. See the "Creating a new project" portion of Chapter 3: Organizing and Managing Your Projects for more information on the AppletViewer.class file.

6) To view the breakpoints you set, select **Show Breakpoints**

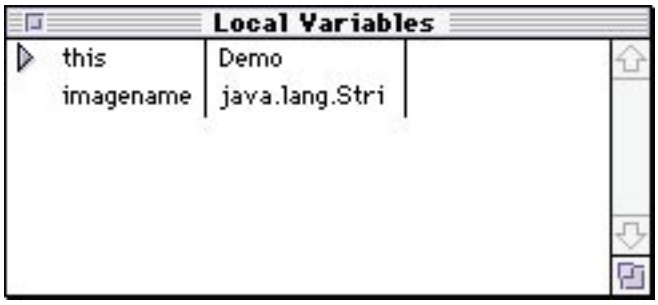
from the **Debug** menu.

7) Select **Go** from either the **Debug** menu or the **Debugger Controls** window. The applet should begin execution and stop at the init method in the Demo.java file.

If you select **Step Over,** you will see the green arrow move down a line at a time in the Demo.java file. Try this 2 or 3 times. When you select **Go** again, the applet will do a little more work, and then display the applet window. As soon as you move your mouse over the applet window, you'll be brought to the mouseMove method in the Demo.java file.

From here you can also see information on the current Java object by using the **Local Variables** menu item under the Windows menu. This command brings up a window that shows the instance variables of the current Java object you are stepping through and their types and values.

At this point the current object should look like:



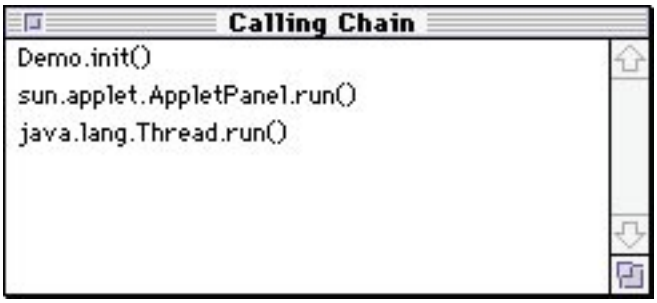
If you double-click on an object or array, you will get a new **Object Inspector** window showing the item's contents. At any point, if you want to continue execution of your applet without interruptions from breakpoints, you can select **Clear All Breakpoints** from the **Debug** menu, then select **Go.**

Now that you've gone through basic steps it takes to create and run an applet, the next chapter shows you how to call your applet from within a web page.

Other important debugger features

The following are some other features you will want to keep in mind when debugging your code:

Current call chain. This displays a list of methods in the call stack. Double-click on one of these methods to go to that area in the code. The **Local Variables** window then updates to where the call chain method is.



Class list. This displays a list of class definitions currently loaded into memory. Selecting a class and unloading it will cause it to be reloaded from disk the next time it is referenced by a new object (i.e., each time a new object of that class is created). The old objects will continue to reference the old class.

This is useful if you have made modifications to your program and wish to test them without restarting the program.

Break on Exceptions. When this option is enabled using the **Debug** menu, when an exception is thrown, the debugger will automatically take you to the point in your code where the exception was thrown, if possible. (For example, if the exception was thrown in a native method, you might not be able to get to that code.)