Eclipse Tutorial

For Introduction to Java Programming By Y. Daniel Liang

This supplement covers the following topics:

- Getting Started with Eclipse
- Choosing a Perspective
- Creating a Project
- Creating a Java Program
- Compiling and Running a Java Program
- Run Java Applications from the Command Line
- Debugging in Eclipse

NOTE: To use this supplement with the text, you may cover Sections 1 - 6 in this supplement after Chapter 1 in the text, cover Section 7 in this supplement after Chapter 2 in the text, and cover Section 8 in this supplement at the beginning of Chapter 15 in the text.

0 Introduction

This tutorial is for students who are currently taking a Java course that uses Eclipse and for Java programmers who want to develop Java projects using Eclipse. Eclipse is an open source supported by IBM.

You can use JDK command line utility to write Java programs. The JDK command line utility consists of a set of separate programs, such as compiler and interpreter, each of which is invoked from a command line. Besides the JDK command line utility, there are more than a dozen Java development tools on the market today, including Borland JBuilder, NetBeans, Sun ONE Studio (a commercial version of NetBeans), Eclipse, and WebGain Visual Café. These tools support an *integrated development environment* (IDE) for rapidly developing Java programs. Editing, compiling, building, debugging, and online help are integrated in one graphical user interface. Using these tools effectively will greatly increase your programming productivity.

This brief tutorial will help you to become familiar with Eclipse. Specifically, you will learn how to create projects, create programs, compile, and run programs. NOTE: Eclipse can run on any platform with a Java Virtual Machine. The screen shots in the tutorial are taken from Windows using Eclipse 3.0. You can download Eclipse from www.eclipse.org.

INSTALLATION NOTE: You must install JDK 1.5 before installing Eclipse. JDK 1.5 can be downloaded from <u>http://java.sun.com/j2se/1.5/download.html</u>. The Windows version of Eclipse 3.0 is contained in a ZIP file named eclipse-SDK-3.0-win32.zip. Unzip the file into c:\. All the files are now contained in c:\eclipse.

1 Getting Started with Eclipse

Assume that you have installed Eclipse files in c:\eclipse. To start Eclipse, double-click on the eclipse icon in the c:\eclipse folder, as shown in Figure 1. The Workspace Launcher window now appears, as shown in Figure 2. Enter c:\smith in the Workspace field and click OK to display the Eclipse UI, as shown in Figure 3. (If the workspace already contains projects, the projects will be displayed in the UI.) Workspace is actually a directory that stores your project files.

| 🔁 eclipse | | | | <u>_ ×</u> |
|---|---|--|-----------------------|----------------|
| <u>File E</u> dit <u>V</u> iew F <u>a</u> vorites | <u>F</u> ools <u>H</u> elp | | | - |
| 🗢 Back 🔹 🔿 👻 🔯 Sear | ch 🔁 Folders 🎯 | $\mathbb{E} \times \mathfrak{O} \equiv V$ | • | |
| Address 🗀 eclipse | | | | |
| eclipse | Configuration features plugins readme | workspace) .eclipseproduct) cpl-v10 eclipse | 🙋 notice 🔋 startup | |
| Type: Application Size: 88.0 KB | | 88.0 KB | y Computer | |

Figure 1

You can start Eclipse by double-clicking the eclipse icon from the eclipse installation directory.

| Workspace Launcher | × |
|---|---|
| Select a workspace | |
| Eclipse Platform stores your projects in a directory called a workspace. Select the workspace directory to use for this session. | |
| Workspace: C:\smith Browse | |
| Use this as the default and do not ask again | |
| OK Cancel | |

The Workspace Launcher lets you choose a directory to store projects.



Figure 3

The Eclipse main window is the command center for the IDE.

2 Choosing a Perspective

A perspective defines the initial set and layout of views in the window. Perspectives control what appears in certain menus and toolbars. For example, a Java perspective contains the views that you would commonly use for editing Java source files, while the Debug perspective contains the views you would use for debugging Java programs. You may switch perspectives, but you need to specify an initial perspective for a workspace.

To create Java programs, set the Java perspective by choosing *Window*, *Open Perspective*, Java from the main menu, as shown in Figure 4. The new UI is shown in Figure 5.

| Resource - Eclipse Platform | | | |
|---------------------------------------|--|-----------------------------------|-----------------|
| File Edit Navigate Search Project Run | Window Help | | |
| Welcome X | New Window | ۵. | (+ ⇒ - ₽ |
| Welcome to Eclipse | Channel Parce A | 🔠 CVS Repository Exploring | - |
| avercome to Echpse | Show View | 🐉 Java | |
| | | 🕵 Java Browsing | |
| | Save Perspective As Reset Perspective | E ⁰ Team Synchronizing | |
| | Close Perspective | Other | |
| | Close All Perspectives | | |
| | Navigation 🕨 | | |
| | Preferences | | _ |
| • | | | <u> </u> |
| | | | |

You need to set a perspective for the workspace.

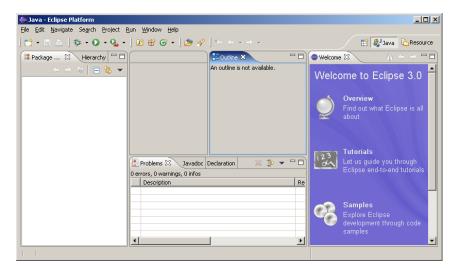


Figure 5

The Eclipse UI is displayed according to the perspective.

3 Creating a Project

To create a project, choose File, New, Project to display the New Project wizard, as shown in Figure 6. Select Java Project and click Next to display New Java Project wizard, as shown in Figure 7. Type myjavaprograms in the Project name field. As you type, the Directory field becomes c:\smith\myjavaprograms. Make sure that you selected the options Create project in workspace and Use project folder as root for sources and class files. Click Finish to create the project.

| 🚝 New Project | | | | × |
|---|----------------|--------|--------|--------|
| Select a wizard Create a Java project | | | | |
| Wizards: Plug-in Project Plug-in Project Plug-in Project Plug-in Project Plug-in Project Plug-in Development Plug-in Development Plug-in Development Plug-in Development | | | | |
| | < <u>B</u> ack | Next > | Einish | Cancel |

Figure 6

The Eclipse UI is displayed according to the perspective.

| 🚝 New Java Project | × |
|--|-----------------------|
| Create a Java project | |
| Create a Java project in the workspace or in an external location. | |
| | |
| Project name: myjavaprograms | |
| | |
| - Location | |
| • Create project in <u>w</u> orkspace | |
| Create project at external location | |
| Directory: C:\smith\myjavaprograms | Browse, |
| | |
| Project layout | |
| • Use project folder as root for sources and class files | |
| C Create separate source and output folders | |
| | |
| | Configure Defaults |
| | |
| | |
| | |
| < <u>B</u> ack <u>N</u> ext > | <u>F</u> inish Cancel |
| | |

Figure 7

The Eclipse UI is displayed according to the perspective.

4 Creating a Program

Now you can create a program in the project by choosing File, New, Class to display the New Java Class wizard, as shown in Figure 8. Type *Welcome* in the Name field. Check the option *public static void main(String[] args)*. Click Finish to generate the template for the source code Welcome.java, as shown in Figure 9.

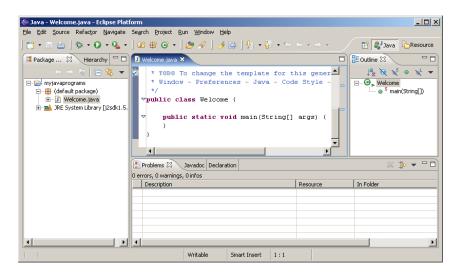
NOTE:

You may use a package by entering a package name in the Package field in Figure 9. Since the source code in the book does not use packages, the Package field is left blank to match the code in the book.

| 🦉 New Java Class | | × |
|------------------------------|---|-------------------------------|
| Java Class | efault package is discouraged. | C |
| Source Fol <u>d</u> er: | myjavaprograms | Browse |
| Pac <u>k</u> age: | (default) | Bro <u>w</u> se |
| Enclosing type: | | Browse |
| Na <u>m</u> e: Modifiers: | Welcome Image: static stati | 1 |
| <u>S</u> uperclass: | java.lang.Object | Brows <u>e</u> |
| Interfaces: | | <u>A</u> dd <u>R</u> emove |
| Which method stubs | would you like to create? public static void main(String[] args) Gonstructors from superclass Inherited abstract methods | |
| | Einish | Cancel |

Figure 8

The New Java Class wizard lets you create a new Java class.



The New Java Class wizard generates the template of Java source code.

Type System.out.println("Welcome to Java"); in the main method.

NOTE: As you type, the code completion assistance may automatically come up to give you suggestions for completing the code. For instance, when you type a dot (.) after System and pause for a second, Eclipse displays a popup menu with suggestions to complete the code, as shown in Figure 10. You can then select the appropriate item from the menu to complete the code.

| 🚝 Java - Welcome.java - Eclipse Plat | |
|---|---|
| <u>File Edit Source Refactor Navigate</u> | Se <u>a</u> rch <u>P</u> roject <u>R</u> un <u>Wi</u> ndow <u>H</u> elp |
|] 📬 • 🔛 🗁] 🏇 • 🕥 • 💁 • | $ \left[\begin{array}{ccc} \underline{0}^{*} & \underline{0}^{*} & \underline{0}^{*} \end{array} \right] \not \underline{0}^{*} & \underline{0}$ |
| Hierarchy | |
| ← → Q E S ▼ | * Window - Preferences - Java - Code Style - */ */ */ public class Welcome (|
| | Stystem. Signature 3 Signature 4 Signature |
| | |
| | Writable Smart Insert 17:16 |

Figure 10

The Code Completion popup menu is automatically displayed to help you complete the code.

5 Compiling and Running a Program

By default, your source code is dynamically compiled as you type. For example, if you forgot to type the semicolon (;) to end the statement, as shown in Figure 11, you will see the red wriggly line in the editor pointing to the error. To run the program, right-click the class in the project to display a context menu, as shown in Figure 12. Choose *Run*, *Java Application* in the context menu to run the class. The output is displayed in the Console pane, as shown in Figure 13.

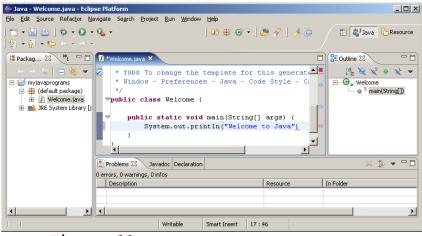


Figure 11

Eclipse dynamically checks syntax errors.

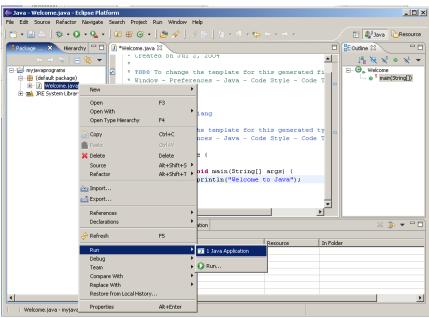


Figure 12

You can run the program from Eclipse.

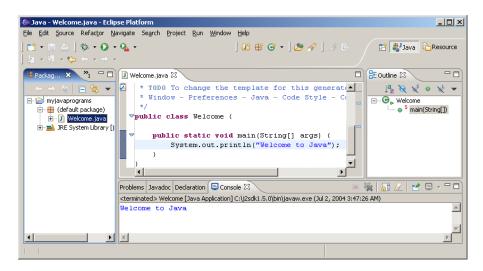


Figure 13

The console pane displays the output to the console.

6 Run Java Applications from the Command Line

You also can run program standalone directly from the operating system. Here are the steps in running the **Welcome** application from the DOS prompt.

 Start a DOS window by clicking the Windows Start button, Programs, MS-DOS Prompt in Windows.
 Type the following commands to set up the proper environment variables for running Java programs in the DOS environment in Windows:

set path=%path%;c:\j2sdk1.5\bin
set classpath=.;%classpath%

- 3. Type cd c:\smith\myjavaprograms to change the directory to c:\smith\myjavaprograms.
- 4. Type **java Welcome** to run the program. A sample run of the output is shown in Figure 14.



Figure 14

You can run the Java program from the DOS prompt using the java command.

NOTE: You can also compile the program using the javac command at the DOS prompt, as shown in Figure 14.

7 Debugging in Eclipse

The debugger utility is integrated in Eclipse. You can pinpoint bugs in your program with the help of the Eclipse debugger without leaving the IDE. The Eclipse debugger enables you to set breakpoints and execute programs line by line. As your program executes, you can watch the values stored in variables, observe which methods are being called, and know what events have occurred in the program.

To demonstrate debugging, Let us use Listing 2.9, ShowCurrentTime.java, to demonstrate debugging. Create a new class named ShowCurrentTime under c:\smith.

7.1 Setting Breakpoints

You can execute a program line by line to trace it, but this is time-consuming if you are debugging a large program. Often, you know that some parts of the program work fine. It makes no sense to trace these parts when you only need to trace the lines of code that are likely to have bugs. In cases of this kind, you can use breakpoints. A breakpoint is a stop sign placed on a line of source code that tells the debugger to pause when this line is encountered. The debugger executes every line until it encounters a breakpoint, so you can trace the part of the program at the breakpoint. Using the breakpoint, you can quickly move over the sections you know work correctly and concentrate on the sections causing problems.

There are several ways to set a breakpoint on a line. One quick way is to click the cutter of the line on which you want to put a breakpoint. You will see the line highlighted, as shown in Figure 15. You also can set breakpoints by choosing *Run*, *Toggle Line Breakpoint*. To remove a breakpoint, simply click the cutter of the line.

As you debug your program, you can set as many breakpoints as you want, and can remove breakpoints at any time during debugging. The project retains the breakpoints you have set when you exit the project. The breakpoints are restored when you reopen it.

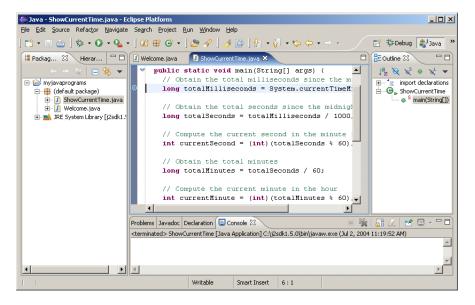


Figure 15

You can set breakpoints in the source code.

7.2 Starting the Debugger

There are several ways to start the debugger. A simple way is shown below:

1. Set a break point at the first statement in the main method in the Source Editor.

2. Right-click on ShowCurrentTime.java in the project pane to display a context menu. Choose Debug, Java Application to start debugging. You will first see the Confirm Perspective Switch dialog, as shown in Figure 16. Click Yes to switch to the Debug perspective. The UI for Debug perspective is shown in Figure 17.



Figure 16

To start debug, Eclipse needs to switch to the Debug perspective.

| 🗲 Debug - ShowCurrentTime.java - Eclipse Platform | |
|--|----------------|
| Eile Edit Source Refactor Navigate Search Project Run Window Help | |
| | Debug 🐉 Java 👋 |
| P Debug X Preakpoints |) 📲 🗖 🗖 🗍 |
| 🕪 🔢 📕 사실 🙀 🚴 😙 🚓 😓 😿 👻 🚥 🔍 args= String[0] (id=11) | |
| - J ShowCurrentTime [Java Application] | |
| E ShowCurrentTime at localhost:1046 | |
| E 🔐 Thread [main] (Suspended (breakpoint at line 6 in ShowCurrentTim | |
| | |
| | - |
| | Þ |
| 🕽 Welcome.java 🗊 ShowCurrentTime.java 🕱 🗖 🗄 Outline 🕅 | -0 |
| public static void main(String[] args) { | 🕅 🗙 🖉 🖌 🖉 |
| // Obtain the total milliseconds since the midnight, Jan 1, 1970 🔄 🕀 👘 import | declarations |
| long totalMilliseconds = System.currentTimeMillis(); B-Op ShowC | urrentTime |
| • • • • • • • • • • • • • • • • • • • | ain(String[]) |
| // Obtain the total seconds since the midnight, Jan 1, 1970 | |
| long totalSeconds = totalMilliseconds / 1000; | |
| 📮 Console 🕴 Tasks 📕 🙀 🚮 🖉 | 2 🛃 🖸 - 🖓 🗋 |
| ShowCurrentTime [Java Application] C:\j2sdk1.5.0\bin\javaw.exe (Jul 2, 2004 11:50:20 AM) | |
| | <u>^</u> |
| | -1 |
| <u> </u> | |
| Writable Smart Insert 6:1 | |

Figure 17

The debugger starts to run ShowCurrentTime.java.

7.3 Controlling Program Execution

The program pauses at the first line in the <u>main</u> method. This line, called the *current execution point*, is highlighted in green. The execution point marks the next line of source code to be executed by the debugger.

When the program pauses at the execution point, you can issue debugging commands to control the execution of the program. You also can inspect or modify the values of variables in the program.

When Eclipse is in the debugging mode, the toolbar buttons for debugging are displayed in the Debug window, as shown in Figure 17. The toolbar button commands also appear in the Run menu (see Figure 18). Here are the commands for controlling program execution:

- **Resume** resumes the execution of a paused program.
- Suspend temporarily stops execution of a program.
- Terminate ends the current debugging session.
- **Step Into** executes a single statement or steps into a method.
- **Step Over** executes a single statement. If the statement contains a call to a method, the entire method is executed without stepping through it.
- **Step Return** executes all the statements in the current method and returns to its caller.
- Run to Line runs the program, starting from the current execution point, and pauses and places the execution point on the line of code containing the cursor, or at a breakpoint.

| Debug - ShowCurrentTime.java - Eclipse Platform | | | |
|---|--|--------------|----------------------------------|
| ile Edit Source Refactor Navigate Search Project | Run Window Help | | 1 |
| 📸 • 🗟 🚊 🛛 🎄 • 💽 • 🎭 • 🛛 🤌 | Toggle Line Breakpoint | Ctrl+Shift+B | 🔛 🏇 Debug 🐉 Java |
| Tebug X | Toggle Method Breakpoint | | 1 👬 🗖 🔻 🗖 |
| ▶ II ■ N % 3. 0 | 🍄 Toggle Watchpoint | | |
| ShowCurrentTime [Java Application] | 🔌 Skip All Breakpoints | | |
| ShowCurrentTime [Java Application] | ^J Add Java Exception Breakpoint. | | |
| 🗄 🔐 Thread [main] (Suspended (breakpoint at line | Add Class Load Breakpoint | | |
| ShowCurrentTime.main(String[]) line: 6 C:\j2sdk1.5.0\bin\javaw.exe (Jul 2, 2004 11:50:2 | Resume | F8 | |
| | Suspend | 10 | |
| ۱ | Terminate | | F |
| 🕖 Welcome.java 🚺 ShowCurrentTime.java 🕱 | | F5 | Outline 🛛 🖓 |
| v public static void main(String[] | Step Into | | La N NS O NL |
| <pre>// Obtain the total millisecor</pre> | 🔿 Step Over | F6 | |
| Iong totalMilliseconds = Syste | _tstep Return | F7 | □ G _▶ ShowCurrentTime |
| <pre>// Obtain the total seconds s:</pre> | ⇒]Run to Line | Ctrl+R. | S main(String[]) |
| long totalSeconds = totalMill: | 😿 Use Step Filters | Shift+F5 | |
| | 🗞 Run Last Launched | Ctrl+F11 | |
| // Compute the current second | S Debug Last Launched | F11 | |
| <pre>int currentSecond = (int)(tots</pre> | | | |
| // Obtain the total minutes | Run History Run As | | |
| <pre>long totalMinutes = totalSecor</pre> | Run | | |
| | Debug History | • | |
| <pre>// Compute the current minute int currentMinute = (int)(tota</pre> | Debug As | • | |
| (Inc) (0000 | Debug | | |
| | R Inspect | Ctrl+Shift+I | |
| Console ಔ Tasks Tasks Tasks Tasks Tasks Tasks | Display | Ctrl+Shift+D | |
| nowcurrenchine (bava Application) C:()250k1.5.0(0In()avaw | C'Execute | Cirl+U | |
| | Step into Selection | Ctrl+F5 | |
| 1 | Q External Tools | | |
| | | ;1 | |

© Copyright Y. Daniel Liang, 2005

The debugging commands appear under the Debug menu.

7.4 Examining and Modifying Variables

Among the most powerful features of an integrated debugger is its capability to examine the values of variables, array items, and objects, or the values of the parameters passed in a method call. You also can modify a variable value if you want to try a new value to continue debugging without restarting the program.

To demonstrate it, choose *Run*, *Step Over* to execute one line in the source code, and you will see the value for <u>totalMilliseconds</u> in the Variables pane, as shown in Figure 19.

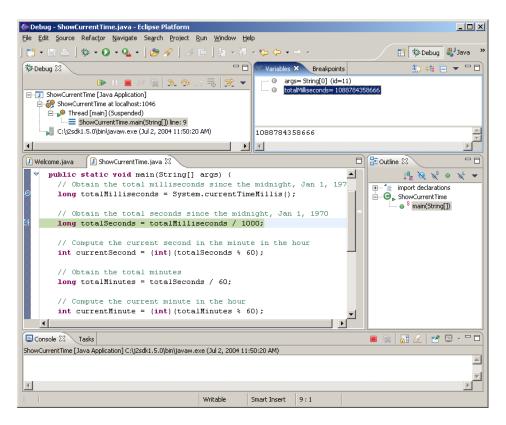


Figure 19

The value for variable <u>totalMilliseconds</u> is displayed in the Variable pane.

To change the value in <u>totalMilliseconds</u>, double-click on <u>totalMilliseconds</u> to display the Set Value dialog box, as shown in Figure 20. You can now set a new value for totalMilliseconds.

| 🗲 Set Value | X |
|--|-----------|
| Enter a new value for totalMilliseconds: | |
| 1088784358666 | × |
| T | F |
| | OK Cancel |

The Set Value dialog box enables you to change the value for a variable.

TIP:

The debugger is an indispensable, powerful tool that boosts your programming productivity. It may take you some time to become familiar with it, but the effort will pay off in the long run.

Note:

The debugger is not only a valuable tool for finding errors, but it is also a valuable pedagogical tool for learning programming.

Note:

After finishing debugging, you may switch to the Java perspective by choosing *Window*, *Open Perspective*, *Java*.

8 Creating and Testing Java Applets

You can create a Java applet in the same way you create a Java application. For example, you can create the WelcomeApplet class in Chapter 14, as shown in Figure 21.

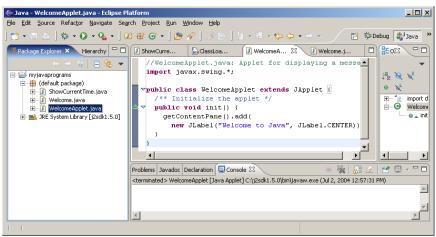


Figure 21

Applets are created in the same way as applications.

To run an applet, choose *Run*, *Run As*, *Java Applet*, as shown in Figure 22. Eclipse automatically creates an HTML file to contain the applet and invokes the appletviewer utility to run the applet, as shown in Figure 23.

| 🚰 Java - WelcomeApplet.java - Eclipse Pl | | | | |
|--|---------------|--|--------------|--|
| File Edit Source Refactor Navigate Sea | | | culu chifu n | E * Dohua #Jaure 3 |
| 📬 • 🔡 👜 🅸 • 💽 • Q₂ • 0 | | Toggle Line Breakpoint Toggle Method Breakpoint | Ctrl+Shift+B | |
| Hierarchy | ShowCur | Toggle Watchpoint | | Welcome.java |
| | //We impo | Skip All Breakpoints | | .ng a message 📥 🛛 👻 |
| myjavaprograms myjavaprograms metric (default package) | τιιφο | 19 Add Java Exception Breakpoint | | $\downarrow^{\mathbf{a}}_{\mathbf{Z}} \aleph \aleph$ |
| | ⊽publ | Add Class Load Breakpoint | | rt { |
| | /* - ⇒⊽ pu | Run Last Launched | Ctrl+F11 | import d |
| | | Tebug Last Launched | F11 | 📖 💿 👗 init |
| | | Run History | • | -:1.CENTER)); |
| |) | Run As | • | 🕎 1 Java Applet |
| | _ | Run | | 🗊 2 Java Application |
| | | Debug History Debug As | • | Ĵ⁰ 3 JUnit Plug-in Test |
| | 4 | Debug | • | Ju 4 JUnit Test |
| | Problems Ja | R Inspect | Ctrl+Shift+I | 9 5 Run-time Workbench |
| | WelcomeApp | Display | Ctrl+Shift+D | 148:00 PM) |
| | | C Execute | Ctrl+U | |
| | | Step into Selection | | |
| | <u> </u> | Q_External Tools | • | <u> </u> |
| | | | | |

Figure 22

Applets are created in the same way as applications.

| 👙 Applet Viewer: WelcomeApplet.clas | s _ 🗆 🗙 |
|-------------------------------------|---------|
| Applet | |
| Welcome to Java | |
| Applet started. | |

Figure 23

The WelcomeApplet program runs from the applet viewer.

$^{\odot}$ Copyright Y. Daniel Liang, 2005

24