

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 and Section 7 in this supplement after Chapter 2 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. The most popular ones are NetBeans and Eclipse. 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.

INSTALLATION NOTE: You must install JDK 1.8 before installing Eclipse. JDK 1.8 can be downloaded from <https://jdk8.java.net/download.html>.

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 need to download the latest version of Eclipse from <http://downloads.eclipse.org/eclipse-java8/2013-05-19/>. The Windows version of Eclipse is contained in a ZIP file. 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.

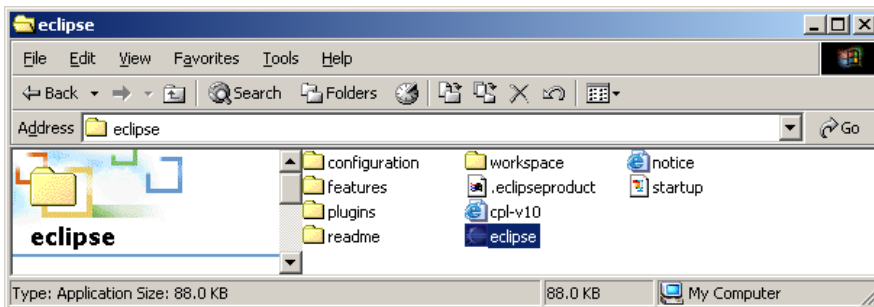


Figure 1

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

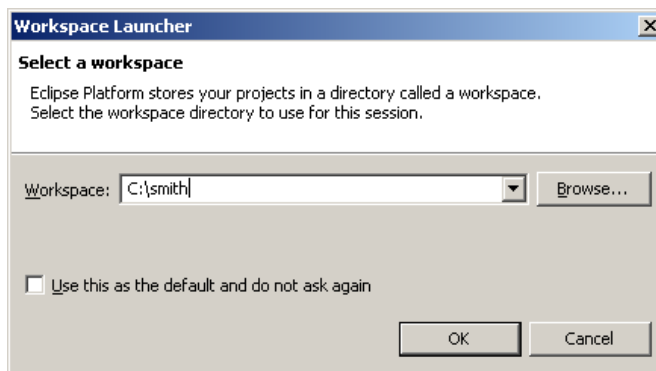


Figure 2

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

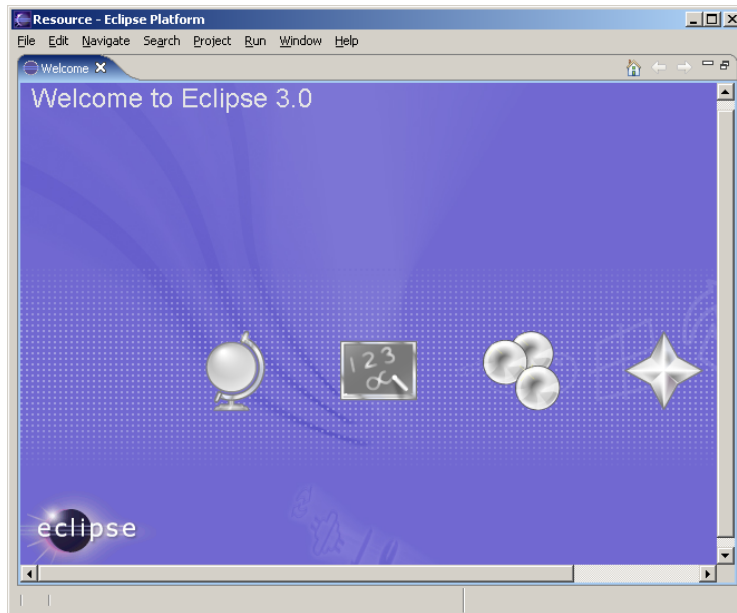


Figure 3

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.

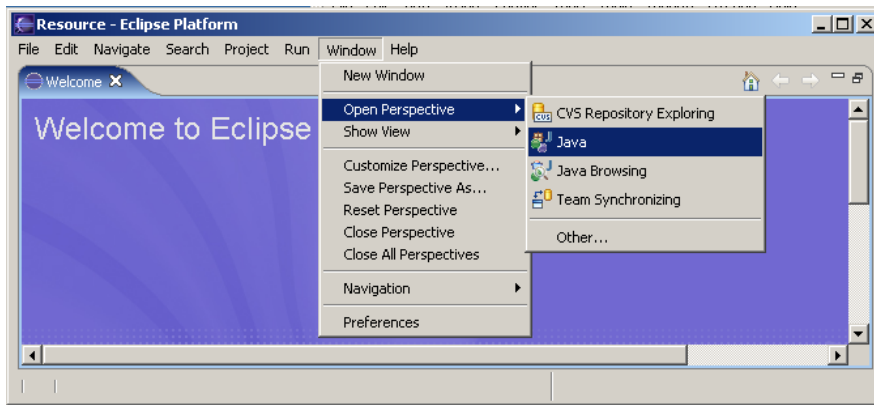


Figure 4

You need to set a perspective for the workspace.

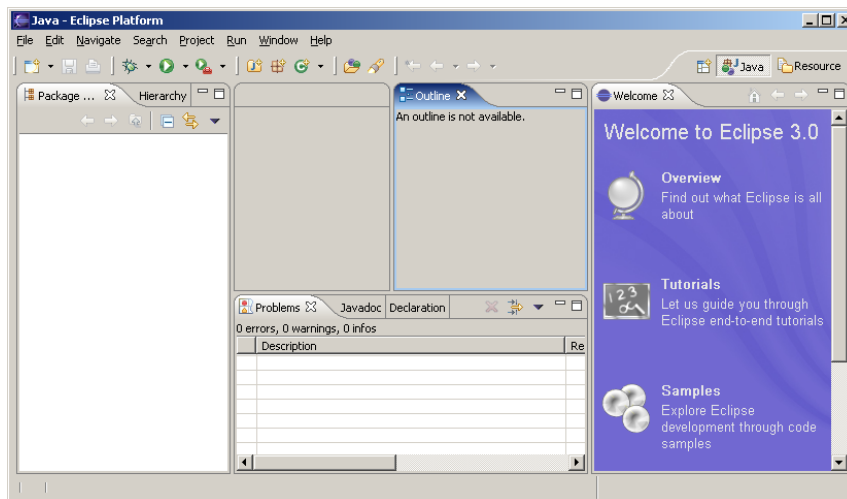


Figure 5

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.

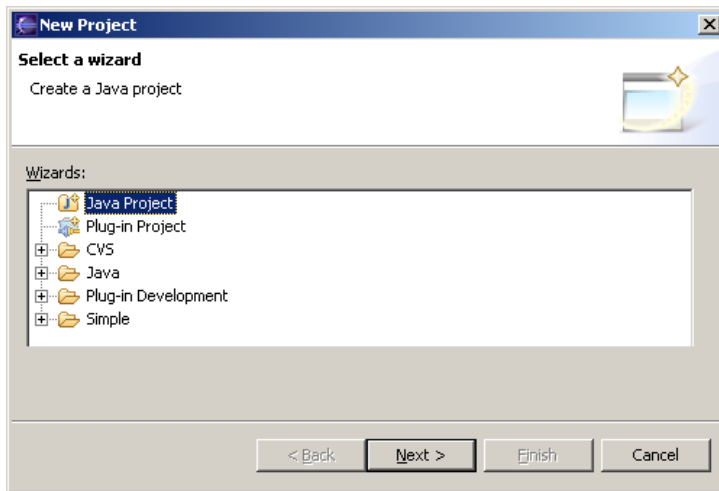


Figure 6

The Eclipse UI is displayed according to the perspective.

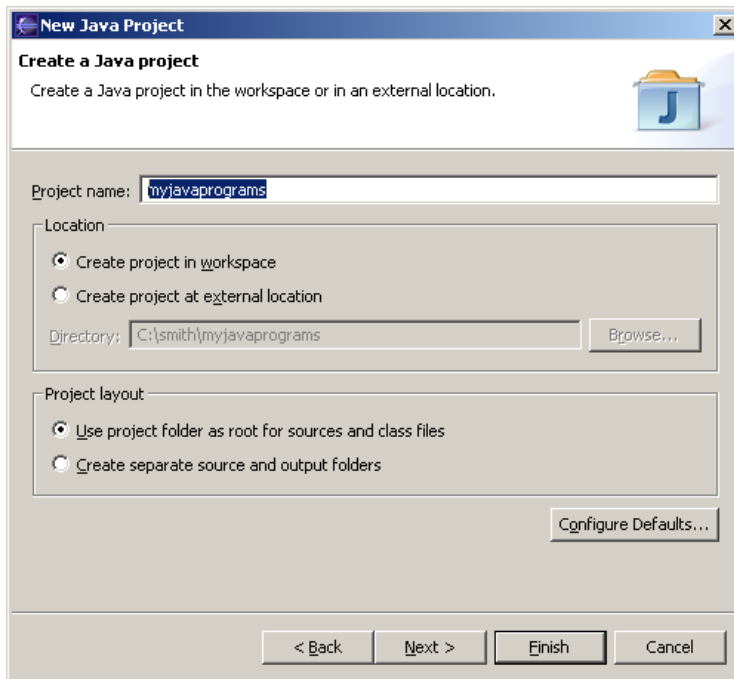


Figure 7

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.

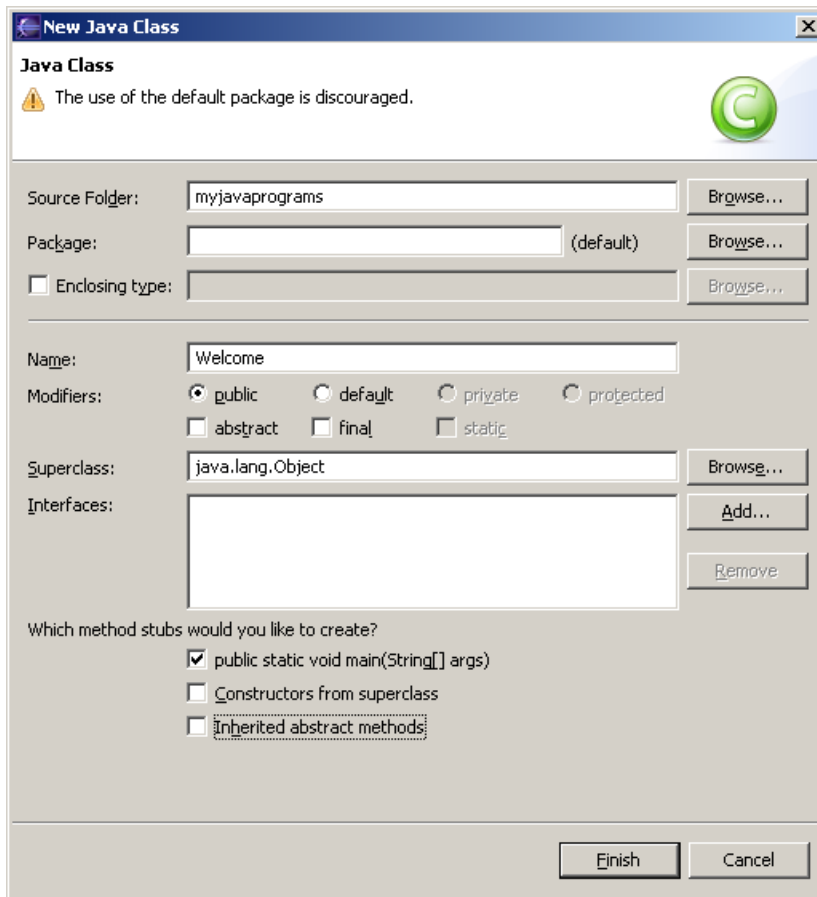


Figure 8

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

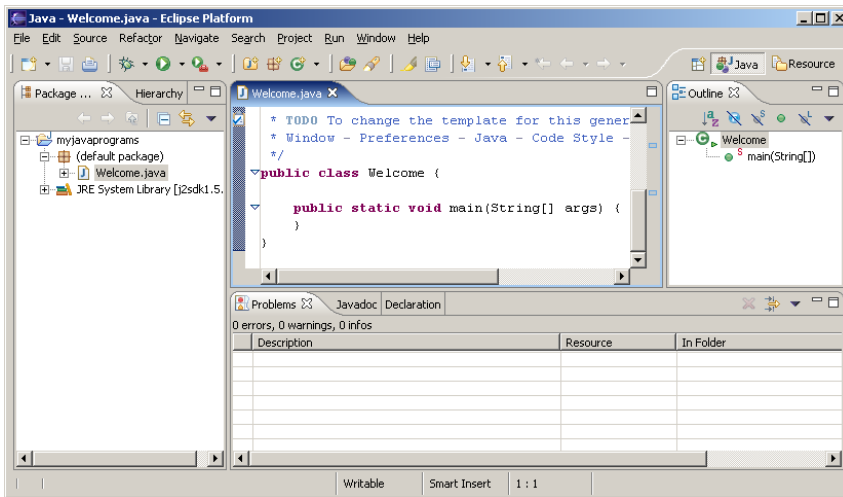


Figure 9

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.

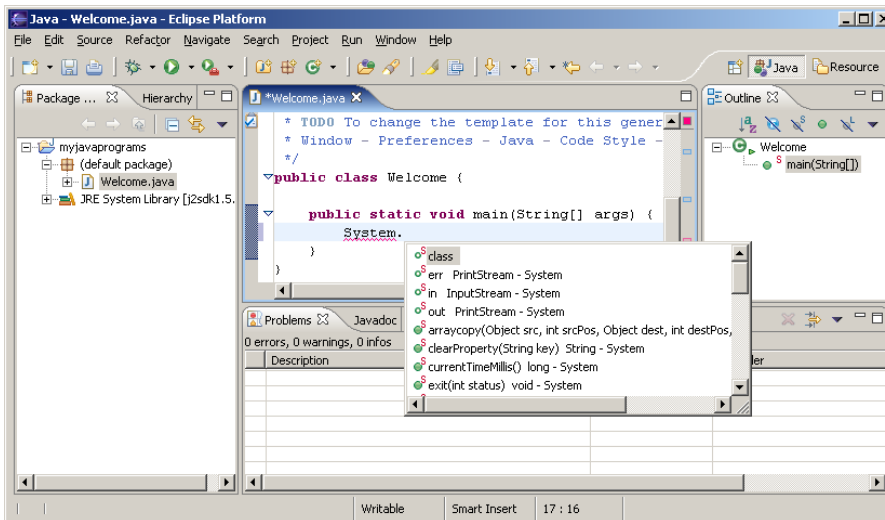


Figure 10

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 wiggly 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.

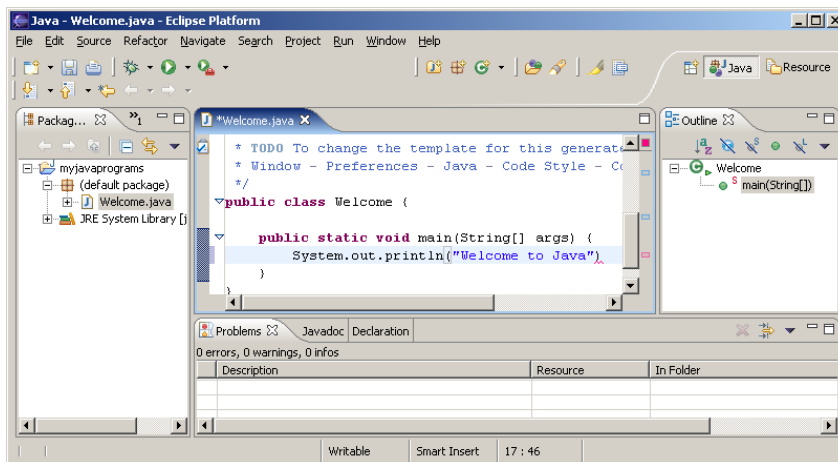


Figure 11

Eclipse dynamically checks syntax errors.

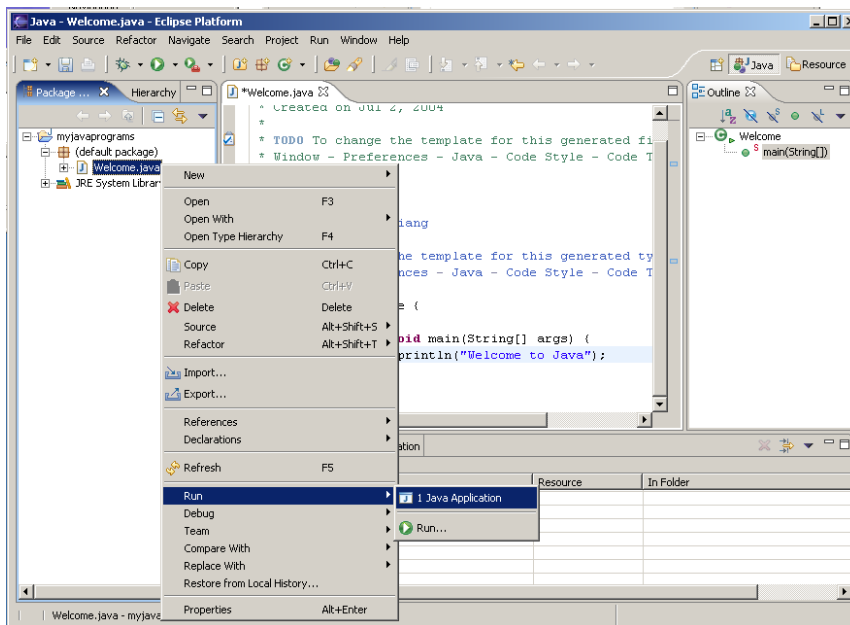


Figure 12

You can run the program from Eclipse.

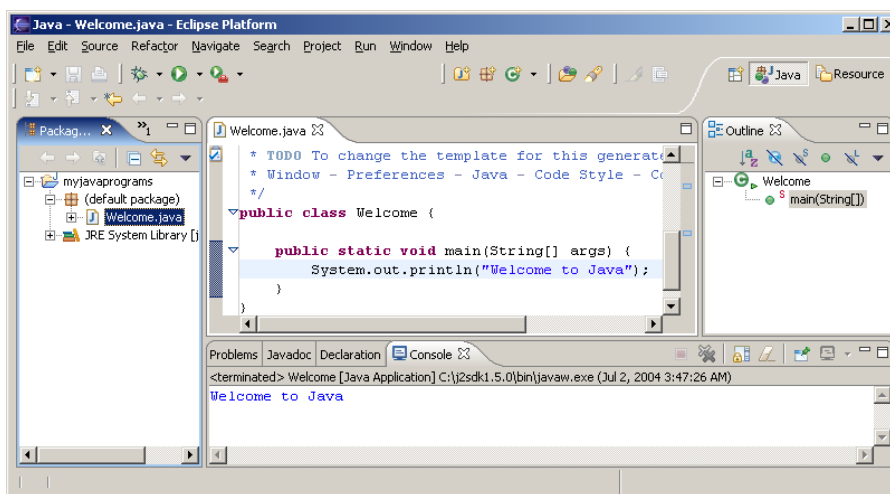


Figure 13

The console pane displays the output to the console.

6 Run Java Applications from the Command Line

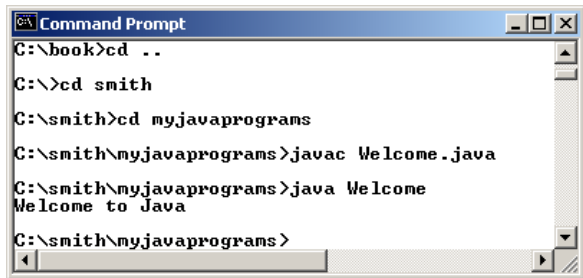
You also can run program standalone directly from the operating system. Here are the steps in running the **Welcome** application from the DOS prompt.

1. Start a DOS window by clicking the Windows Start button, Programs, MS-DOS Prompt in Windows.
2. 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.



```
Command Prompt
C:\book>cd ..
C:\>cd smith
C:\smith>cd myjavaprograms
C:\smith\myjavaprograms>javac Welcome.java
C:\smith\myjavaprograms>java Welcome
Welcome to Java
C:\smith\myjavaprograms>
```

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.

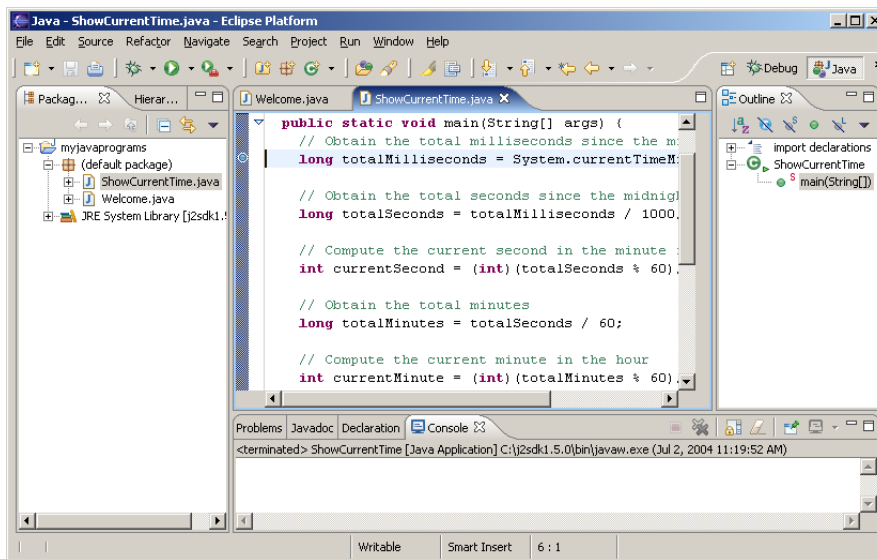


Figure 15

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 breakpoint 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.

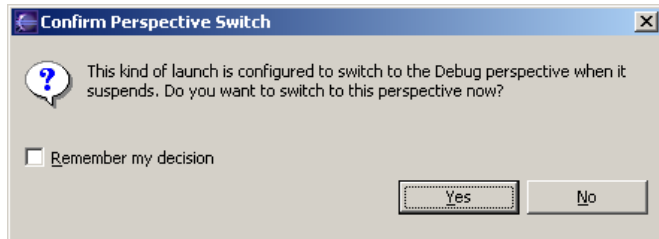


Figure 16

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

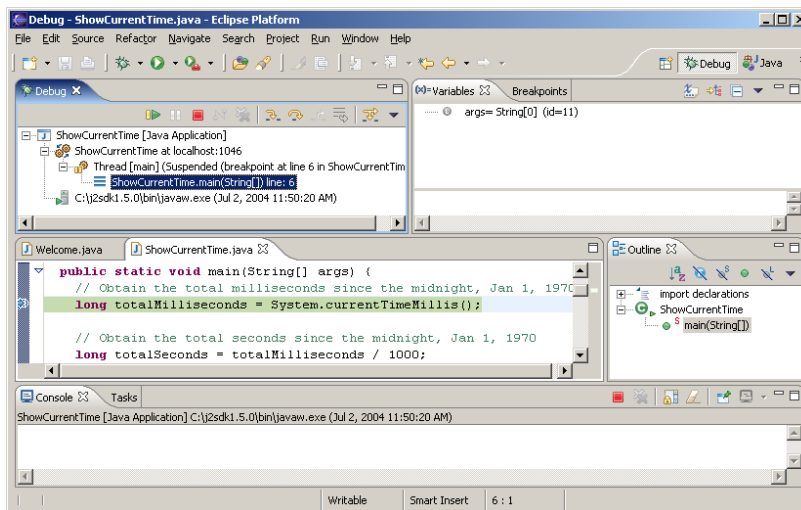


Figure 17

The debugger starts to run ShowCurrentTime.java.

7.3 Controlling Program Execution

The program pauses at the first line in the `main` 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.

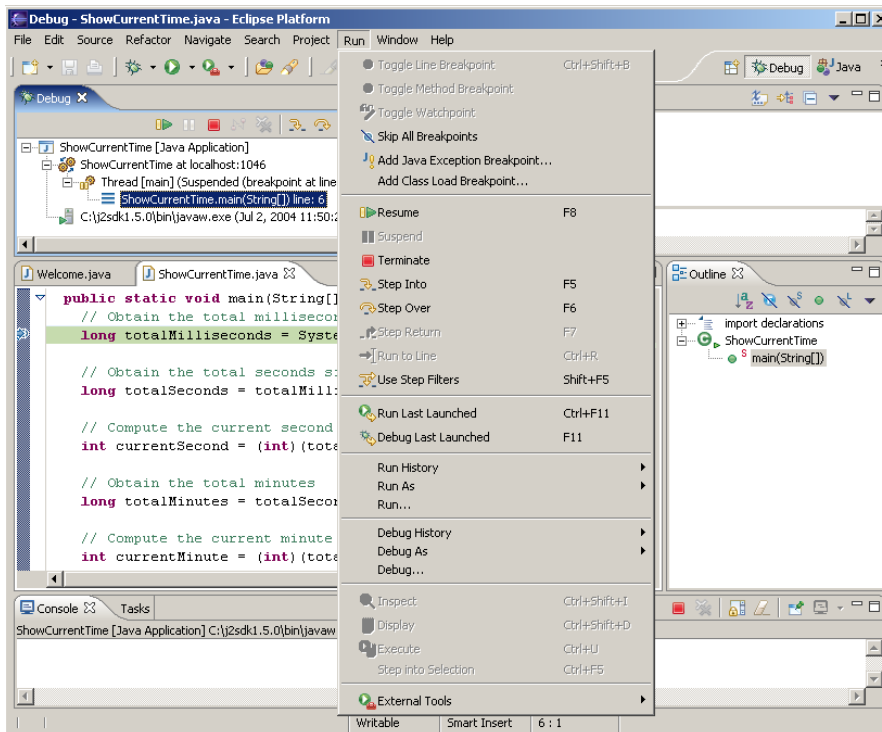


Figure 18

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 totalMilliseconds in the Variables pane, as shown in Figure 19.

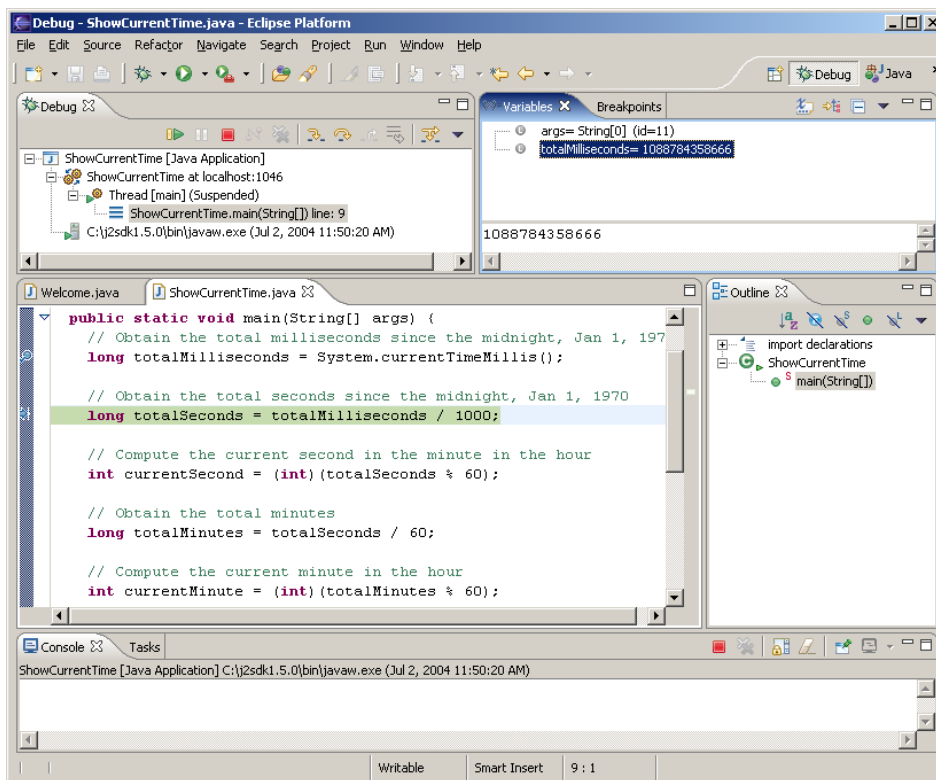


Figure 19

The value for variable totalMilliseconds is displayed in the Variable pane.

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

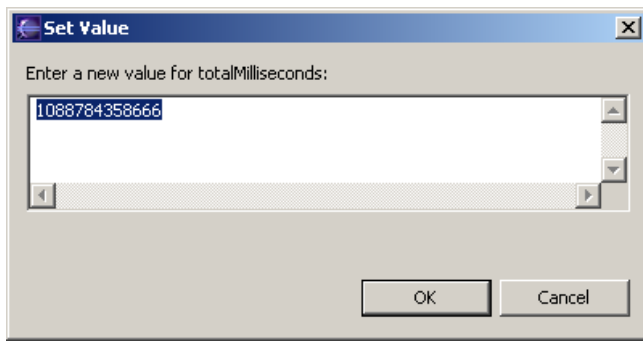


Figure 20

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*.