

CHAPTER**5****Debugging in Forte**

Debugging is finding errors (i.e. *bugs*) in a program and correcting them. Programming errors can be separated into syntax errors, runtime errors, and logic errors. Runtime errors that cause the program to abort are reported by the Java runtime system. Syntax errors are detected and reported by the compiler. In general, errors of these two kinds are easy to locate and fix. Therefore, debugging usually means to find logic errors.

Logic errors result in incorrect output or cause a program to terminate unexpectedly. To find logic errors, you can *hand trace* the program (that is, catch errors by reading the program) or insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility. This chapter introduces debugging in Forte.

5.1 General Debugging Techniques

Debugger utilities let you follow the execution of a program. They differ from one system to another, but all of them support most of the following helpful features:

- o Executing a single statement at a time[md]The debugger allows you to execute one statement at a time so that you can see the effect of each statement.
- o Tracing into or stepping over a method[md]If a method is being executed, you can ask the debugger to enter it and execute one statement at a time, or you can ask it to step over the entire method. You should step over the entire method if you know the method works. For example, always step over system-supplied methods like System.out.println().
- o Setting breakpoints[md]You can also set a breakpoint at a specific statement. When your program reaches a breakpoint, it pauses and displays the line with the breakpoint. You can set as many breakpoints as you want. Breakpoints are especially useful when you know where your programming error begins. You can set a breakpoint at that line and have the program execute until it reaches the breakpoint.
- o Displaying variables[md]The debugger lets you select several variables and display their values. As you

trace through a program, the contents of the variables are continuously updated.

- o Using call stacks[md]The debugger lets you trace all of the method calls and lists all pending methods. This feature is helpful when you need to see a large picture of the program execution flow.
- o Modifying variables[md]Some debuggers enable you to modify the value of a variable when debugging. This is convenient when you want to test a program with different samples but do not want to leave the debugger.

The debugger utility is integrated in Forte. You can pinpoint bugs in your program with the help of the Forte debugger without leaving the IDE. The Forte 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.

5.2 Starting the Debugger

To demonstrate debugging, this chapter uses the following class example. The class named SelectionSort contains the selectionSort method. Suppose a mistake is made in the selectionSort method, as shown in the following code listing at the highlighted line:

```
// SelectionSort.java: Sort numbers using selection sort
public class SelectionSort {
    /** Main method */
    public static void main(String[] args) {
        // Initialize the list
        double[] myList = {5.0, 4.4, 1.9, 2.9, 3.4, 3.5};

        // Print the original list
        System.out.println("My list before sort is: ");
        printList(myList);

        // Sort the list
        selectionSort(myList);

        // Print the sorted list
        System.out.println();
        System.out.println("My list after sort is: ");
        printList(myList);
    }

    /** The method for printing numbers */
    static void printList(double[] list) {
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
        System.out.println();
    }

    /** The method for sorting the numbers */
    static void selectionSort(double[] list) {
        double currentMax;
        int currentMaxIndex;

        for (int i = list.length - 1; i >= 1; i--) {
            // Find the maximum in the list[0..i]
            currentMax = list[i];
            currentMaxIndex = i;

            for (int j = i - 1; j >= 0; j--) {
```


interface, as shown in Figure 5.2. The toolbar button commands also appear in the Debug menu (see Figure 5.3). Here are the commands for controlling program execution:

Start begins to debug the current program.

Finish ends the current debugging session.

Attach opens a dialog box in which you can connect the debugger to an application on another virtual machine. This is useful for remote debugging in distributed systems.

Pause temporarily stops execution of a program.

Run to Cursor runs the program, starting from the current execution point, and pauses and places the execution point either on the line of code containing the cursor or at a breakpoint.

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 Into executes a single statement or steps into a method.

Step Out executes all the statements in the current method and returns to its caller.

Continue resumes execution of a paused program.

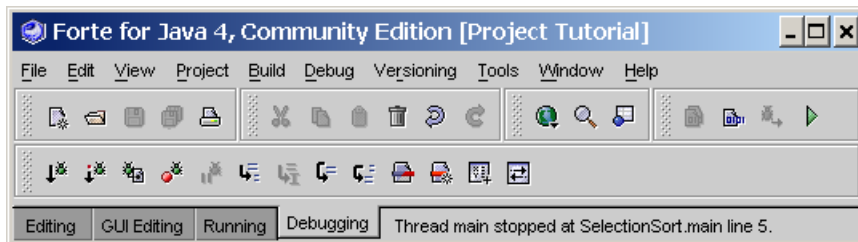


Figure 5.2

The debugging toolbar buttons are displayed.

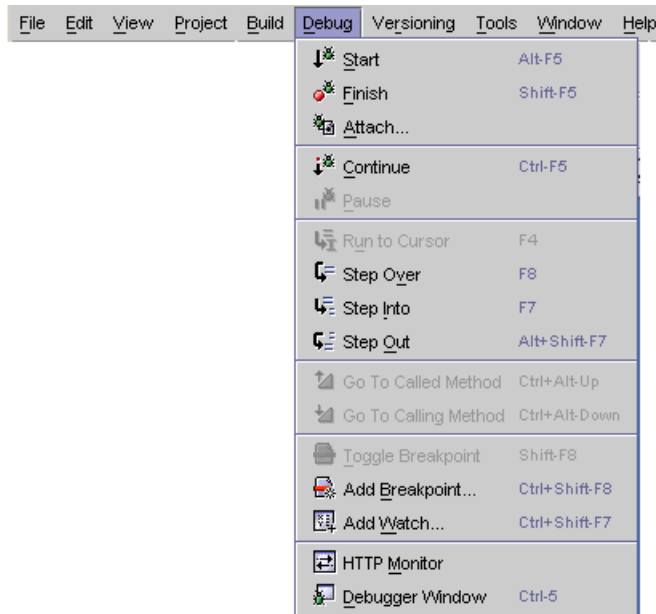


Figure 5.3

The debugging commands appear under the Debug menu.

5.9.3 The Debugger Window

The Debugger window can be displayed by choosing *View, Debugger Window*, or *Debug, Debugger Window*. The Debugger window can have split panes for sessions, breakpoints, threads, call stacks, watches, variables, classes, and properties, as shown in Figure 5.4. These panes can be selected or deselected by clicking a toolbar button in the Debugger window. The Sessions pane lists the current debug sessions. The Breakpoints pane lists all the breakpoints you have set. The Threads pane lists threads and thread groups in the current debugging process. The CallStack pane lists the method calls that the process has made since it began running. The Watches pane lists the variables and expressions that are under continuous watch. The Variables pane lists changes in variables and expressions as the program is being debugged. The Classes pane lists all the classes that have been loaded by the process being debugged.

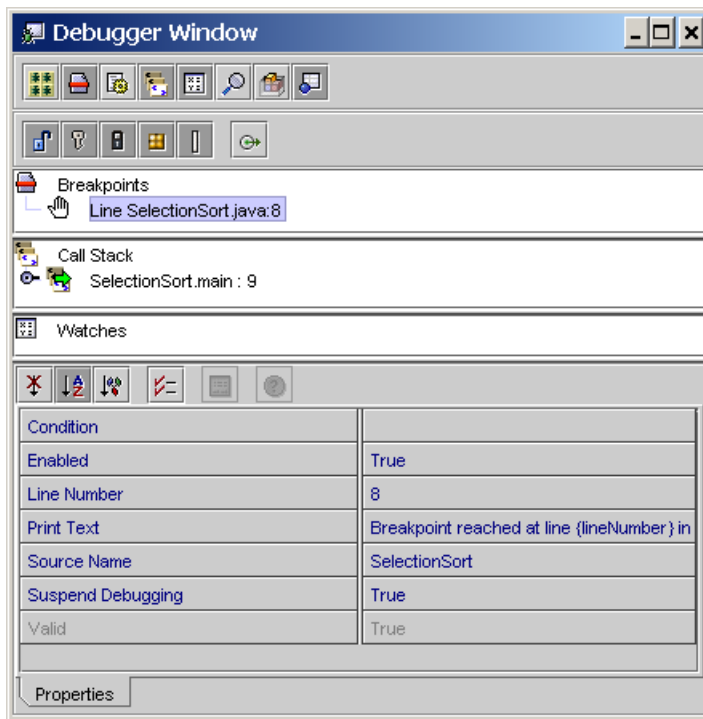


Figure 5.4

The Debugger window displays debugging information for the program.

5.9.4 Examining and Modifying Data Values

Among the most powerful features of an integrated debugger is its capability to reveal current data values and enable programmers to modify values during debugging. You can 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.

Forte provides the Add Watch command to enable you to add variables to the Watches tab in the Debugger window. You can then inspect and modify the values of variables.

5.9.4.1 The Add Watch Command

The Add Watch command adds variables to the Watches tab so that you can watch the changing values of variables while debugging. To add the variable `myList` in the `SelectionSortWithError` to the Watch view, perform the following steps:

1. Suppose the execution point is currently at the first line in the `main` method. Highlight `myList` in the Source Editor and right-click the mouse to reveal a context menu.

2. Choose *Add Watch* in the context menu to bring up a dialog box, as shown in Figure 5.5. Click *OK* to add myList to the Watch list.
3. Choose the *Watches* tab in the Debugger window. The variable along with its content is shown in Figure 5.6.
4. Choose *Debug, Step Over* to observe the changing value of myList in the *Watches* tab.

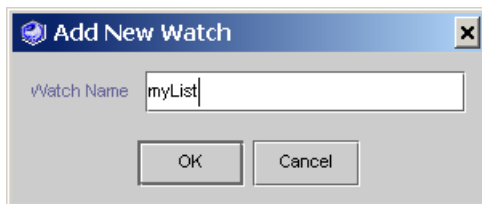


Figure 5.5

The Add New Watch dialog box enables you to add a variable or an expression to the Watch view.

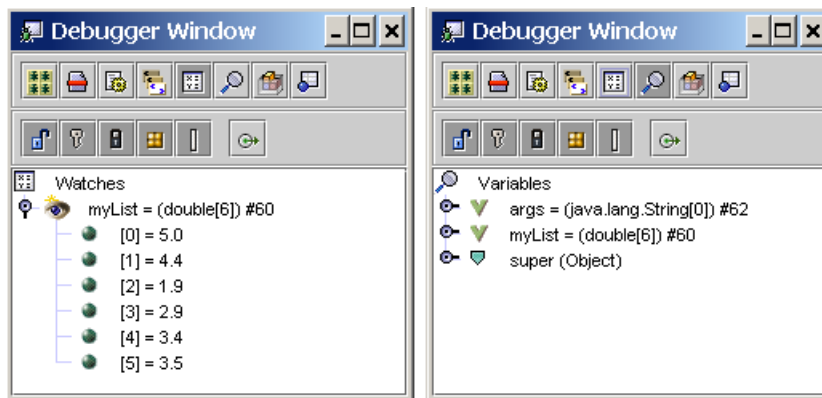


Figure 5.6

*The variable myList was added to the *Watches* tab.*

Forte NOTE:

You can add expressions such as i > 0 to the *Watches* tab from the *Add New Watch* dialog box.

Forte NOTE:

You can also view the values of the variables in the *Variables* tab.

5.9.4.2 Modifying Variables

You can modify variables from the Watches tab or the Variables tab. The following steps show how to modify the value of a variable:

1. Select myList[0] in the left-hand pane of the Watches tab.
2. Modify the value field to change the value for myList[0] in the right-hand pane.

5.9.5 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. You can then trace the part of the program at the breakpoint, quickly moving over the sections that work correctly and concentrating on those causing problems.

There are several ways to set a breakpoint. One quick way is to use the control key. To set a breakpoint, move the cursor to the line where you want to set a breakpoint and press CTRL+F8. To remove a breakpoint, move the cursor to the line with a breakpoint and press CTRL+F8.

When debugging a program, you can set as many breakpoints as you want, and can remove breakpoints at any time during debugging. The project retains the breakpoints when you exit the project. They are restored when you reopen it.

5.9.6 Debugging SelectionSort

Use the debugger to uncover the bugs in SelectionSortWithError by performing the following steps:

1. Since you know there is nothing wrong in the code before invoking the selectionSort method, you can skip the lines before calling selectionSort in the main method by setting a breakpoint at the line for selectionSort(myList) in the main method.
2. Choose SelectionSortWithError.java in the project pane. Choose Debug, Start to debug the program. The debugger executes every line until it reaches the breakpoint.
3. Choose Debug, Step Into to debug the selectionSort method.

4. Add list and currentMax to the Watches tab in the Debugger window to monitor changes of values in these variables. Check whether list has the correct initial values before the method starts (see Figure 5.7).
5. The selectionSort method places the largest number at the end of the list after an iteration of the outer for loop. Run through the iteration at full speed by setting the cursor at the if statement for swapping numbers, then choose Debug, Run to Cursor.
6. Examine the value of currentMax in the Watches tab, as shown in Figure 5.8. Clearly, currentMax is not getting the correct value. The correct value should be 5.0. Assign list[j] to currentMax, not list[i].
7. Change list[i] to list[j] and choose Debug, Start. A dialog box prompts you to recompile and restart debugging. Click Finish and Start to close the current session and start a new session.

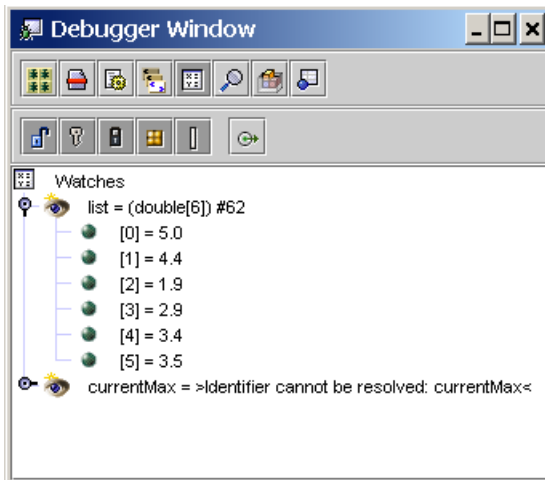


Figure 5.7

You can add the variable list and currentMax to the Watches tab.

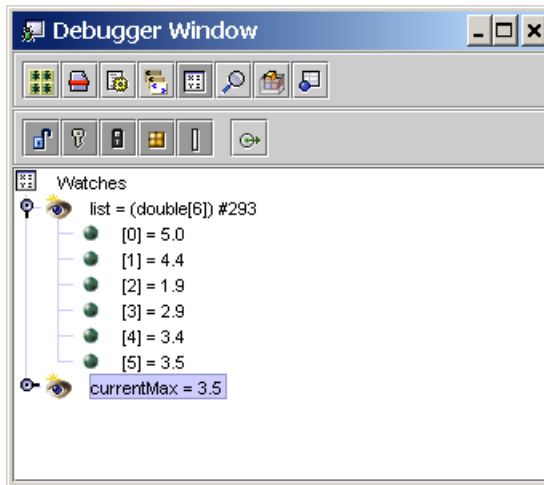


Figure 5.8

currentMax should be 5.0, but it is 3.5 as shown in the Watches tab.

Forte TIP:

You can show current value of a variable by pointing the mouse at the variable in the Source Editor. The value is displayed as a ToopTip above the mouse pointer.

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.