Debugging Java Programs with CodeWarrior for CS100 and CS211

Computer Science, Cornell University

Fall 1999

CodeWarrior includes an interactive debugger that you can use to examine your program as it executes. This handout gives an overview of the debugger and how to use it effectively. Some of the information in this handout may not make much sense until later in the course. Skim over unfamiliar material now and read it again later at the appropriate time.

1 Perspective

A good interactive debugger, which allows you to run your program a few statements at a time and observe the values of variables, is an invaluable tool for finding problems and verifying that a program works.

But a debugger is no substitute for thinking. The best way to eliminate errors (bugs) is to not create them in the first place. Time spent carefully designing a program is more than repaid in time saved during testing and debugging. Before you start writing detailed code, be sure you understand the problem and the algorithms you plan to use to solve it. After you've written the code, carefully check it (proofreading, tracing, etc.) before you try running it.

Don't rush to get something on the computer. You may be tempted to type in the first thing that comes to mind and start tinkering with it, because it sometimes feels like no progress has been made until the machine is involved. But if you hack before you know what you are doing, it will take much longer to get a correct program and the code will be much worse.

Rather than tinkering with your code, randomly changing things in the hope that bugs will goaway, get away from the computer and think.

Take advantage of all error-detecting features provided by CodeWarrior. Turn on any options that you can find to generate warning messages. Use any available software tools to look for potential bugs. Don't waste your time on problems the computer could have caught for you.

Once you have carefully designed and typed your program, your task is to verify that it works as expected. The debugger can be a great help with this, allowing you to stop execution at "interesting" places and check that variables have their expected values, that expected output has been produced, and that nothing unexpected has happened.

2 Using the Debugger

Select Project|Enable Debugger. (On a Mac, this changes command Run in menu Project to Debug.) Then select Project|Debug to have the debugger execute your program.

The debugger's *program* window (with an uninformative title like AppClasses.jar) opens automatically when you Debug (instead of Run) a program. Another useful window is the *class browser* window, which you can create with View|Class Browser on PC. (On a Mac, you can create with Window|New Class Browser.)

The program window contains two panes on top and one on the bottom. A list of active methods is in the upper-left panel. The bottom panel displays the source file containing the method whose name is selected in the list and the upper-right panel displays its local variables. An arrow appears to the left of the next statement to be executed in each active method. Initially, execution is paused at the first statement of method `main`.

The browser window has four panes. (There are also four tabs; in CS100, we care about only Java and Methods, not Properties and Events.) It can be used to view any class in the program, not just those containing currently active methods. The left-most pane contains a list of classes in the program. When a class name is selected, its methods are shown listed in the top-left pane and its class and instance variables are listed in the top-right pane. Select a method name to view its source code in the bottom pane.

Arrange the debugger's windows on the screen so they don't overlap other windows, like the input/output console window, that you want to see while debugging. You can change the size of the debugger's windows by dragging the lower right corner, as usual. To adjust the size of individual panes in the windows, use the mouse to drag the vertical or horizontal bars between them.

When the debugger begins executing your program, it usually pauses at the first statement. You can cause execution to proceed by selecting Project|Run (this assumes you have first selected Debug). Select Run twice if the first selection only brings the program window forward without actually executing the program. Alternatively, you can execute the program one statement at a time, as discussed in Section 2.3.

2.1 Breakpoints

Before you run your program, you can set *breakpoints*: places in the code where you want execution to pause so you can examine the situation before continuing. Breakpoints can be set in both the program and browser windows. In the source panes of these windows, there are dashes to the left of most statements, indicating places where breakpoints can be set. (This is a good reason to put individual statements on separate lines. Breakpoints can be placed only at the beginning of a line, not in the middle.)

To set a breakpoint, click one of the dashes. It will turn into a small red dot. When the execution reaches an active breakpoint, the program pauses and control returns to the debugger.

The *breakpoints window* contains a list of all the breakpoints in the program. If it's not on the screen, you can select it from View|Breakpoints.

You can temporarily disable a breakpoint without removing it entirely by clicking the red dot next to it in the breakpoints window. The dot changes to a circle. The breakpoint still exists in the program, but execution won't pause there. This is useful, for example, if you want to stop at a breakpoint the first time it is reached but not after. To reactivate a breakpoint, click the circle in the left column of the breakpoint window to cause the red dot to reappear.

A breakpoint may have a *condition* attached to it – a logical expression (e.g. `sum >= 17`). When execution reaches a breakpoint with a condition, it pauses only if the breakpoint is active and the condition is true. Conditions are entered in the breakpoints window to the right of the affected breakpoint.

You can view the location of a breakpoint in the source program by double clicking it in the breakpoints window. The browser window will jump to that location.

To permanently remove a breakpoint from the program, click the red dot next to the statement in the program window or symbol window (not the breakpoints window). The dot will change back to a dash to indicate that the breakpoint is gone.

Note: The Pro 3 debugger for Windows does not always pause properly at the beginning of method main when it begins executing a program. If this happens, set a breakpoint at the start of the `main` method. ■

2.2 Stopping Execution of a Program

When using the debugger, you can cause execution of the program to pause by selecting Debug|Break. To terminate execution of the program, select Debug|Kill, or try one of the following if you're on a Mac.

Macintosh. To stop a program you are debugging (perhaps because it is in an infinite loop), type control-command-/. This should return control to the debugger. The program window will show the location in the program where execution is paused, and you can use the debugger to appraise the situation.

If control-command-/ doesn't work, try forcing termination by pressing Option-Command-Esc. If you want to continue debugging, you will have to start from the beginning.

Occasionally a problem will cause the machine to lock up or freeze. If neither of the above steps salvages the situation, you will have to restart the machine. Don't do this unless you have to – all unsaved work in open applications will be lost.

2.3 Incremental Execution

Once a program is stopped, you may want to look around and run it slowly, a few statements at a time. The commands in menu Debug do this.

The *step* commands run the program one statement at a time. Step Over and Step Into both execute the next statement in the program. If that statement calls a method, Step Over executes the entire method call in a single step. Step Into moves into the body of the method so you can trace its execution. Use Step Into to examine your own methods and Step Over for methods supplied by us or CodeWarrior.

Step Out completes execution of the current method and stops at the place where the method was called.

Restart resumes execution until another breakpoint is reached or until the program terminates.

Break terminates execution and returns control to the debugger. Execution can be resumed after executing any desired debugger commands.

Kill terminates execution of the program. This is useful if you want to start the program over from the beginning without waiting for it to finish.

2.4 Examining a Program

When the program is stopped, you can examine the values of variables, check whether certain conditions are true or false, evaluate expressions, etc.

2.4.1 The Call Stack

The *stack* in the upper-left corner of the program window displays a list of currently active methods. The name of the currently executing method appears at the bottom of this list, the name of the method that called it is just above, and so forth. For example, if method `main` called `sort_list`, which called `move_smallest_to_front`, which called `move_element`, the call stack would be:

```
main
sort_list
move_smallest_to_front
move_element
```

Additional methods may be listed above `main`. These methods are either part of CodeWarrior or part of the operating system. Normally you will be interested only in the ones from `main` down.

You can view the source code and local variables of any method in the call stack by selecting its name.

2.4.2 Local Variables

The upper right corner of the program window shows the local and global variables of the method currently selected in the call stack. The current value of each variable appears to the right of its name.

The values of simple variables, with types like `int`, `double`, and `char`, are displayed to the right of the variable name. For arrays and structures, the memory location of the variable is shown and there is a disclosure square (▶) to the left of the variable name. Click this button to see the components of the variable (array elements or structure fields). More complex data structures may contain fields that can be further expanded by clicking a disclosure square.

Note: Uninitialized variables may not appear. ■

To view the value of a variable in a separate window, double click its name or select View Variable from menu Data.

Other commands in menu Data control how information about variables is displayed.

Show Types displays the type of each variable along with its value.

View As... allows you to set the type according to which a variable is displayed. This is primarily useful for machine level debugging.

View Array displays an array in a separate window. Option-double-clicking a variable name is a shortcut for this command.

To change the value of a variable during debugging, double-click its current value and enter a new one.

Warning: This changes the value of the variable only for the current execution of the program. To make a permanent change, you *must* change the original program. ■

2.4.3 Expressions

You can evaluate expressions in the *expressions* window. Expressions may include program variables or Java operators, but not method calls.

There are two ways to enter expressions in this window. Menu command Data|Copy to Expression copies any selected expression from the program window to the expressions window. To enter a new expression select Data|New Expression, type the expression and press Return.

To modify an expression, double-click it and make the desired changes. Hit Return to display its value.

2.5 When You're Done

Use CodeWarrior to make any needed changes. When you select Project|Debug again the debugger will resume control of the program.

To run the program without the debugger, select Project|Disable Debugger. To resume debugging, select Project|Enable Debugger again.