Lab Exercise - Debugging

Introduction

The NetBeans IDE, available on our lab computers and downloadable for free from Sun, includes as one of its tools a symbolic debugger. When applied to debuggers, the word “symbolic” typically means that the debugger allows you to access your programs at the source code level rather than at the assembly language level. For most types of debugging, this is sufficient (and in most cases more desirable). Most debuggers allow the following types of activity:

• Single step execution
• Inspection of variable values
• Setting breakpoints – specifying a line number or function body where execution should stop and allow other debugger functions to be used
• Conditional breakpoints – specify a condition on the program’s variables which, when true, interrupts execution and allows debugger functions to be used
• Call stack tracing – show the series of function calls with parameter values that led to the current situation

Laboratory Tasks

1. Create a new folder in an appropriate place called “debugging” for this laboratory.
2. In NetBeans, create a new project called “Set” in your debugging folder. The type of project should be “Java Application.” Uncheck the “Create main class” option before proceeding. (If you leave this box checked, you are forced to use the Java package mechanism. This is not in itself a bad thing, but it adds a level of complexity that we don’t need here.)
3. On the class assignments page, look for this assignment and download the file debugging.jar, placing it in the “src” subfolder that NetBeans created in debugging/Set.
4. Extract the contents of the jar archive by typing

   ```
   jar xvf debugging.jar
   ```

   You will have to open a command window and navigate to your src folder using the “cd” (change directory) command before typing the jar command. Note that if you need to change drives in the command window, you do this just by typing the drive designator (such as U:) and hitting return.
5. In the top left-hand pane of NetBeans, find your new project and use the mouse to navigate to Set->Source packages-><default package> and be sure you see the files from the jar archive there. (If you don’t see your project, be sure the “Projects” tab is selected for this pane.)
6. In order for the debugger to be fully useful, the .java files need to be compiled for debugging. This is turned on by default in NetBeans, but just so you know where this is done, right-click the name of your project in the top left pane and pick “Properties” from the pop-up menu. Select the “compiling” option under “Build” and note the state of the “Generate debugging info” check box. Once you are done debugging, you will want to uncheck this box and do a clean rebuild to reduce the size of your executable.

7. Build the program. NetBeans may ask you to identify the main class for the project. It is testSet.

8. The source code is, as the name suggests, a Set data structure. It uses a binary search tree to store the elements for the set, and provides the normal operations you would expect, like union, intersection, set difference, etc. The testSet class is a test harness that exercises the various methods of the Set class. There is a bug in this software, which you should be able to spot by reading through the output when you run testSet.

Your job is to use the debugging capabilities of NetBeans to track down the cause of this error. Below you will find a summary of how to use various debugging features that may come in handy as you do this. You should turn in a modified version of whatever file you need to modify to fix the bug, WITH YOUR NAME WRITTEN ON IT, and the changes you made clearly marked.

**Setting/Clearing Breakpoints**

The most useful feature of debuggers, as you may already know, is that they let you execute your program one line at a time, watching the changes that the code causes to the values of variables. The handiest way to do this is by setting a “breakpoint,” or identifying the line of code where you would like the execution of the program to pause so that you can take control of the process. In NetBeans, you do this by just clicking the mouse to the left of the line of code where you would like execution to stop. If line numbers are turned on, clicking on the line number will do; if not, clicking in the gray area along the left edge of the editor window works. You should see a small pink box appear in this margin area. Clicking on the pink box turns off the breakpoint.

**Program Execution**

You can control execution of the program either from the “Run” menu, with function keys, or through toolbar buttons just above the editor window. In any case, there are several things you’ll want to be able to do.

- Execute the program - cause the program to run up to the first breakpoint that is encountered. (Run->Debug main project, F5, or toolbar button with pink box and green right-facing triangle).
• Step over - execute the currently highlighted statement, “stepping over” any method calls that it makes. (i.e. execute the entire method and stop at the statement after the currently highlighted one.) (Run->Step over, F8, blue toolbar button with a curved arrow on the left side.)

• Step into - execute the currently highlighted statement, but if a method call is made, “step into” the method and pause at the first executable statement of the method. (Run->Step into, F7, blue toolbar button with an arrow pointing down to the middle of the button.

• Continue - stop single-stepping and resume normal execution until the next breakpoint is encountered. (Run->Continue, Control-F5, toolbar button that looks a bit like a “Play” button on a music player.)

• Quit - terminate the debugger session. (Run->Finish debugger session, Shift+F5, brown toolbar button with an “x”).

There are a couple of other program execution controls whose use should be pretty self-evident.

Inspecting Program State

The other useful thing that symbolic debuggers allow us to do is look at the values of our variables and trace how they change as the program executes. In Netbeans, this happens in a pane that pops up below the source code when you begin a debugging session. There are three tabbed panes labeled “Watches,” “Call stack,” and “Local Variables.” If it isn’t already selected, click on the “Local Variables” tab. As you step through variable declaration statements in the method, new entries corresponding to the variables will appear in this list, along with information about their type and contents. If what has been declared is an object reference, a small “+” appears in a box to the left of the reference name, allowing you to expand the variable to look at its fields. In this way, you can follow chains of linked structures. You will always see “this” in the list of variables, which is the implicit self-reference to the object that was used to call the method being debugged. By expanding “this,” you can look at the values of the data fields of the current object. Note that this will, for example, let you get to the root data field of the binary search tree, and by recursively expanding left and right links, to see the entire contents of the tree. When the value of a variable or field was changed by the most recent step of the computation, the value is highlighted in red.

Other capabilities

The NetBeans debugger is fairly powerful, with features to allow debugging of applications with multiple threads of execution, web applications, etc. We will not need most of those features for our work this semester.