Debugging Compression

Lab 5

Wednesday, February 23, 2011

CSC 062: Spring, 2011

As usual, you are welcome to work with another student on this lab. Please work with someone that you have not worked with before. The startup code is available in `/common/cs/cs-62/labs/lab5`.

Eclipse and Debugging

Eclipse is a good example of a professional interactive development environment (IDE) that provides good support in finding and correcting errors. If you have any compile-time errors in your program, a red "X" will appear next to the file name, both in the Package Explorer and in the editing pane. An error message will also appear in the "Problems" pane, at the bottom of the window. If you click on the error message, the editing pane will scroll to the line with the error, which will have an "X" on the left margin. If you hold the mouse over the "X" in the left margin, an error message will eventually appear. If it provides a suggestion for a fix in a second pop-up window, you can usually double click on the suggestion and it will perform it for you.

If you fix the error, the "X" will turn gray, and it will disappear altogether when you save the file. However, while your program may no longer have compile-time errors, it may have run-time errors.

When you run GridTest with Eclipse, it will crash with a null pointer error if you click in the middle of the grid. This will be indicated by the appearance of multiple lines in red in the Console pane in the lower right corner. Scroll to the top of the Console pane and you will see that you got a `NullPointerException`. The line immediately below that message will tell you that the error occurred while executing method `currentValue` of `CurDoublyLinkedList`. If you click on that line, it will open the file in the edit pane and show that line of `CurDoublyLinkedList`. That line is:

```
return current.value();
```

suggesting that the value of `current` was null at that point.

The next line in the Console pane shows that the error occurred during the execution of method `find` from class `CompressedTable`. If you click on this, it will take you to that method. Notice that the line selected is one that calls the `getCurrent` method. That is, the error occurred when method `find` called method `currentValue`, which crashed when evaluating `current.value()`.

Similarly the next line in the console shows this occurred during the execution of `getInfo` in `CompressedTable`. We can trace this back further to method `paint` in `DrawingPanel`, but after that we get into system calls that aren't very meaningful.

Running the Debugger

Identifying where the program blew up is helpful, but we need more information. Why was `current` null? What did the list look like at that point? In this particular case, the error is pretty trivial, though it occurred far away from the methods where it actually blew up. The error was in method `first` of `CurDoublyLinkedList`, where you will find the body has been erroneously written as current = null, rather than current = head. Please fix this so that we can go on to the next problem.

If you now run the program, you'll discover some rather weird behavior. Most of the time when you click somewhere, you'll get the result you expect. However if you click to the right and below all of the places that have color changes, then all of the other changes disappear, showing only the last click.

Your job in this lab is to figure out the problem using Eclipse's debugger.

Quit the current run of the program and start over. Click once and then hit the "Display list" button. Do that a few times and notice what happens to the list. You should see that the list is out of order and only some of those in the list actually show up on the screen. Something is clearly going wrong.

The first thing we do is to set one or more "breakpoints" in the program. Double-click in the gray margin next to one of the first three lines in `updateInfo` in `CompressedTable`. If you hit it just right, a small blue circle appears. If you mess up, the editor might close the method and put a "+" next to its name. If it does

that, click on the "+" to open it again, and try again to double-click in the gray margin. If you double-click again, it will erase it. However, leave it in and let's open up the debugger.

If you are running the program in the debugger, the program will stop at that point in the program. Now you are ready to run the debugger.

Make sure "GridTest.java" is selected and then go to the Run menu and select "Debug as..." and "Application". (To run debug again, you need only click on the picture of the bug to the left of the running person above the Package Explorer pane.) The window will take a bit longer to appear than when running the program. Click on one of the middle squares of the grid and wait for the debugger to come forward. It may ask you if you wish to go into debug perspective. If so, say yes.

When the debugger window comes forward you will see a somewhat more complex display. The upper left pane (labeled "Debug") shows all of the threads running in your program. There will be several shown there, but only one will be selected. It is the thread corresponding to the execution of your program.

The pane below the Debug pane shows the line of your program that was executing when it hit the breakpoint. You can scroll in this window as usual or bring up other files from the tabs.

The pane in the upper right corner of the window should be labeled "Variables" (if not, click the Variables tab at the top of the pane). It shows all of the active variables in your program at this point. For example it should show `row`, `col`, `newInfo`, and `optimizeBefore`. If you click on the triangle next to `newInfo` you can see that it is an object with instance variables `cs`, `alpha`, etc. Moreover the values of those instance variables will be shown (e.g., `row` and `col` will depend on where you click). If a variable represents an object, you will be able to click on the triangle next to it to see the values of its instance variables. (Unfortunately, the info for colors seems to be pretty difficult to decipher.) If you don't want a variable expanded out, just click on the triangle next to it again, and it will fold up to the previous form.

At the top of the list is the word `this`. If you click on the triangle next to it, you will see all of the instance variables of the object executing the code. In this case, you will see all of the instance variables of the compressed table. Click on `tableInfo` to get information on the current state of the doubly linked list. Please write on a piece of paper (to be turned in later) the values of `count` and `current` (i.e., its id number) as well as the positions (row and col only) of the elements of the doubly linked list. (I'll let you figure out how to find out this information.)

While it won't be very interesting at this point, the icons to the right of the word "Debug" on the top of the Debug pane allow you to control the execution of your program. If you click on the green triangle, the program will start executing again. Stopping only if it hits another breakpoint or terminates. If the program is executing, clicking on the icon consisting of two tall narrow yellow rectangles will cause it to pause. Clicking on the red square will terminate your program.

The four yellow arrows allow step by step execution of your program. The first one (turning down) allows you to step into the method called in the highlighted line. If you don't want to see the details of that method's execution, but instead go to the next line of the method shown, click on the second arrow (which goes up and then down) to step over that line. Finally, the third arrow (going up and to the right) will finish executing that method and pop you out to the method that called it. We will ignore the fourth one.

For the rest of this lab, I would like you to determine what has happened to screw up the list when the user clicks below and to the right of the last square changed. In particular, look at the method `updateInfo` of `CompressedTable`. Depending on where the new element is added, it will call either method `handleEndOfList` or `handleNotAtEnd`. In each case, the method `addAfterCurrent(...)` will be called. Please write down a picture of the list, including head, tail, and current after each call to `addAfterCurrent`.

Don't keep running it when the linked list gets too long. Start over and try again to see what is going on. Pay particular attention to the values of the instance variables of `CurDoublyLinkedList`: `head`, `tail`, and `current`. Write these values down and the entire list each time the list changes.

It may be helpful to note that each object used in your program is given a unique id number at run-time. Use the information gathered from the debugger to explain where the first error occurs in the program. (You may need to run the program several times to see where the error occurs, we each have a tendency to assume things are all right, when an error has actually already occurred!). Finally, please explain how the error can be corrected.

Incidentally, you can go back and forth between the debug view and the usual "Java" view of Eclipse by

clicking on icons in the top right corner of the window. Clicking on the icon with a "J Java" on it takes you to the Java view, while clicking on the bug icon with "Debug" will take you to the debug view. Another handy shortcut is that double-clicking on the tab for a file in the edit pane will expand the edit pane to fill the window. Double-clicking it again will reduce it to the original size. This can be handy if you want to read more for editing and then reduce it so you can see the output from running the program.

What to hand in

Please hand in the info requested (including pictures of the linked list at different stages), as well as the explanation of how the error can be corrected. Please hand this in either at the end of lab or at the beginning of class on Friday.