Dr. Emily Hill
CSCI 151

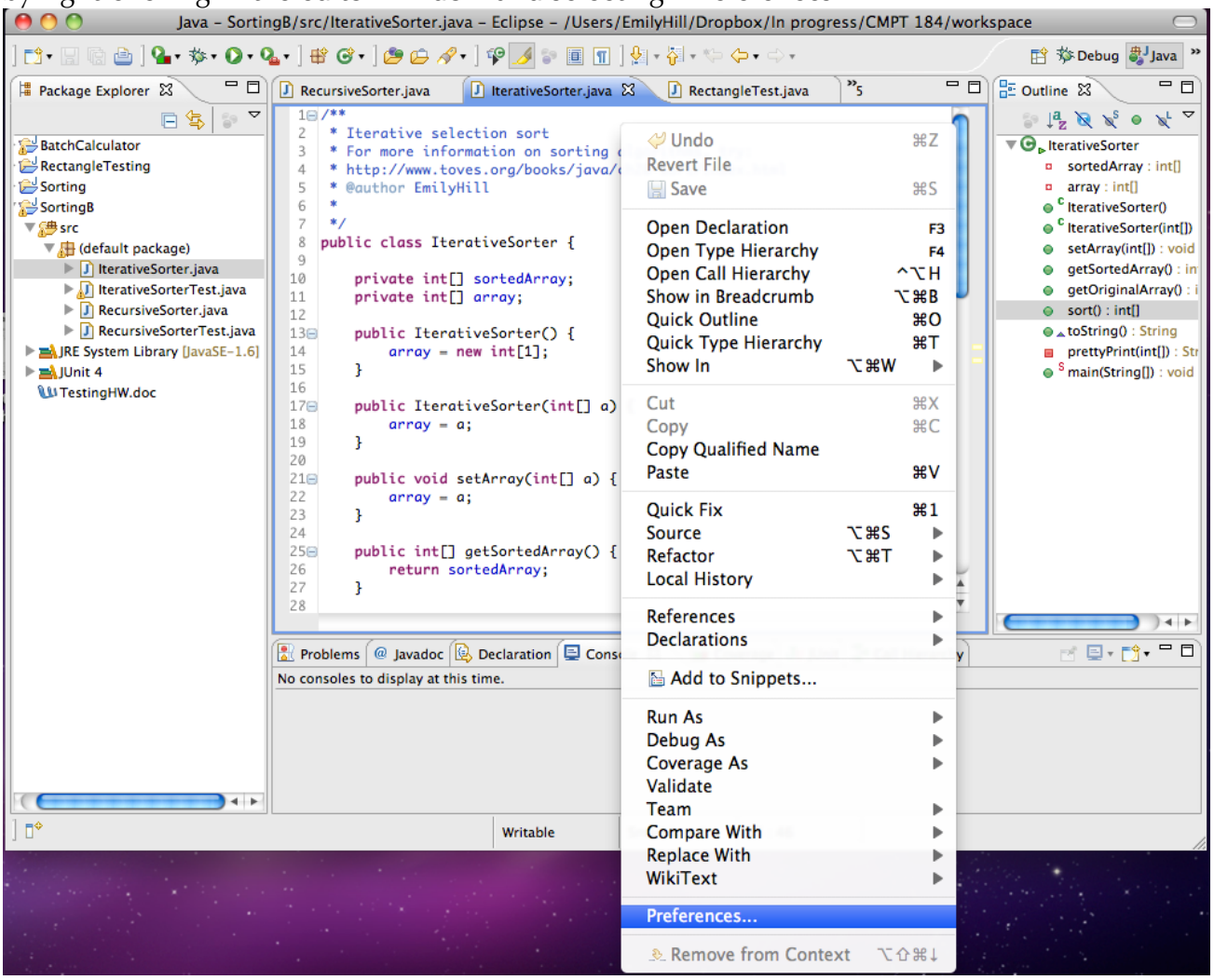
TUTORIAL: ECLIPSE DEBUGGING DEMO

This tutorial assumes you have downloaded the SortingB.zip file from BB and imported the SortingB project into Eclipse. To import the project from the zip file by selecting File > Import and then General > Existing Projects into Workspace. Hit "Next", then "Select archive file," browse to SortingB.zip, and hit "Finish".

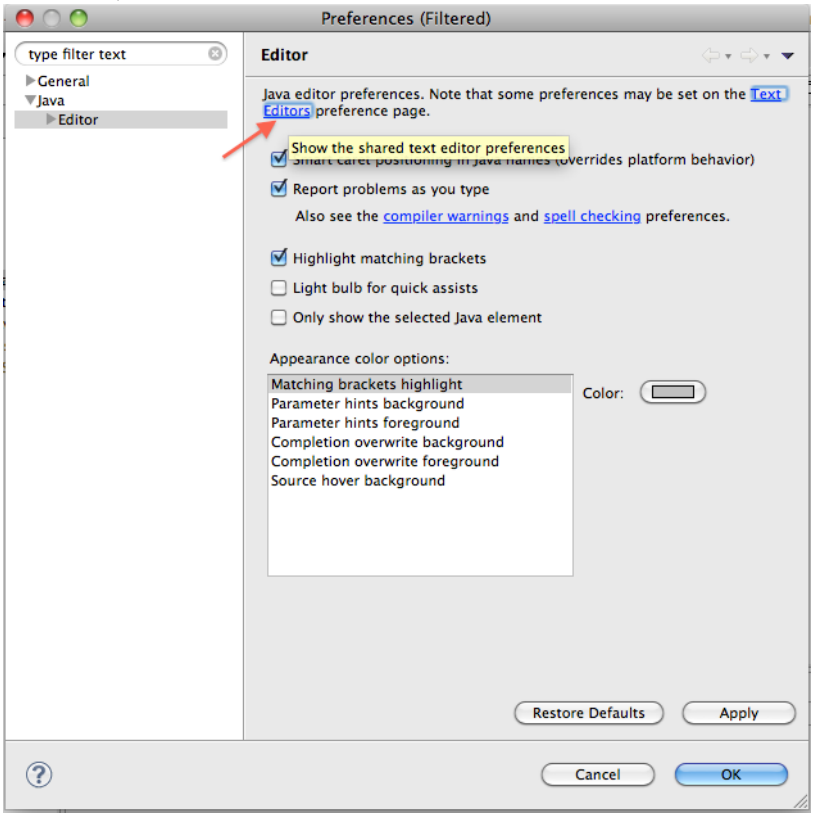
The steps below will show you how to use the Eclipse debugger to step through an execution of `IterativeSorter`.

Displaying Line Numbers

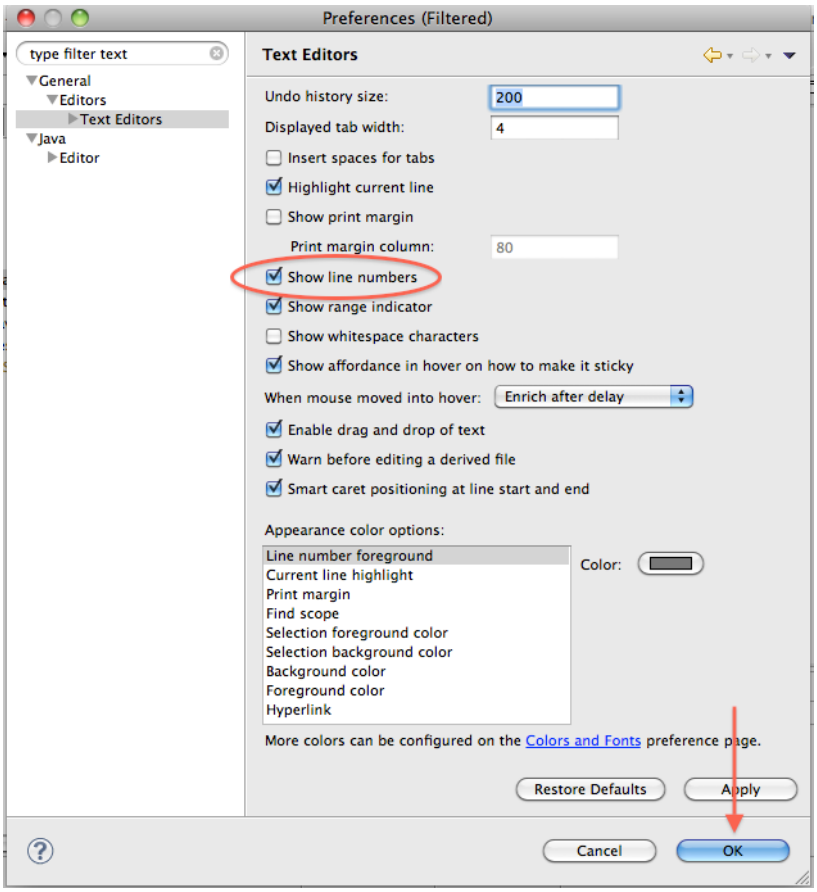
1. Before we begin, it is helpful to have Eclipse display line numbers. Turn on line numbers by right-clicking in the editor window and selecting "Preferences"



2. Next, select “Text Editors”

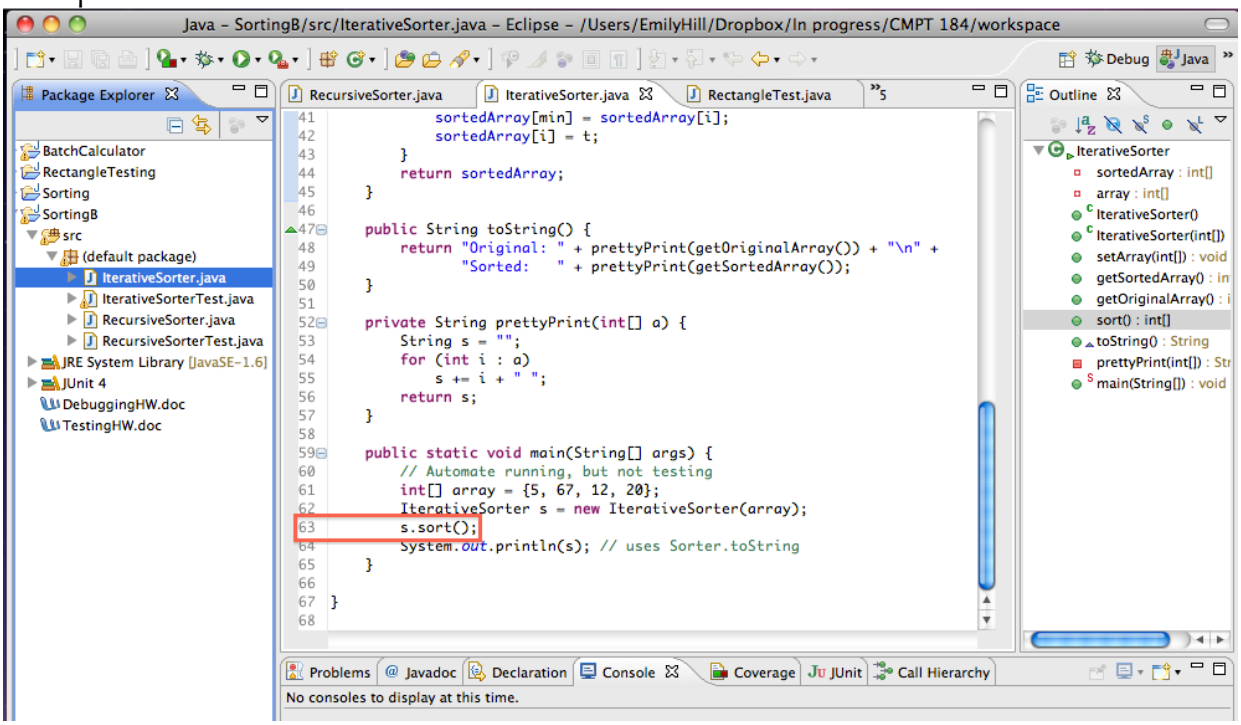


3. Make sure the check box next to “Show line numbers” is selected and hit “OK”

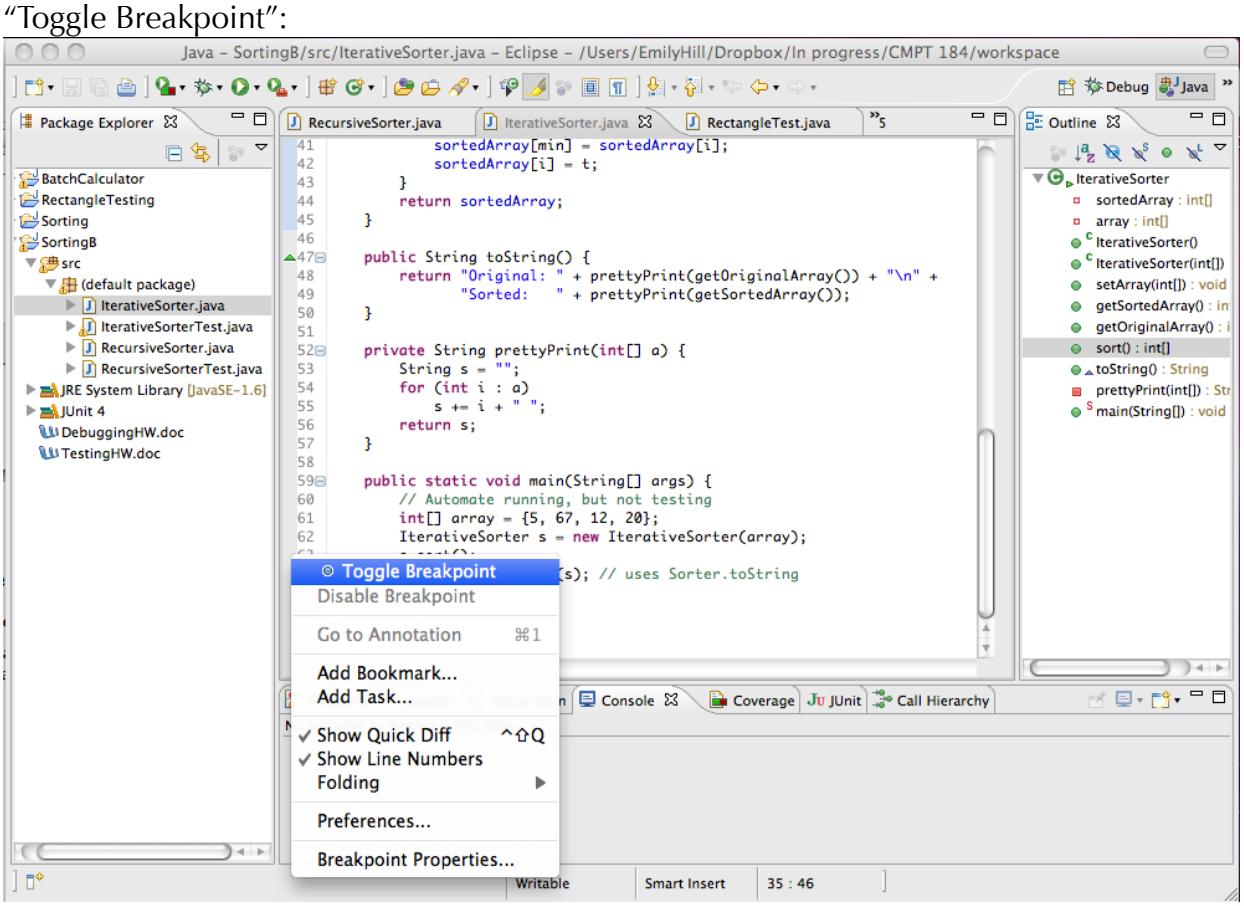


Debugging Demo

1. Open the `IterativeSorter` class and scroll down to its `main` method



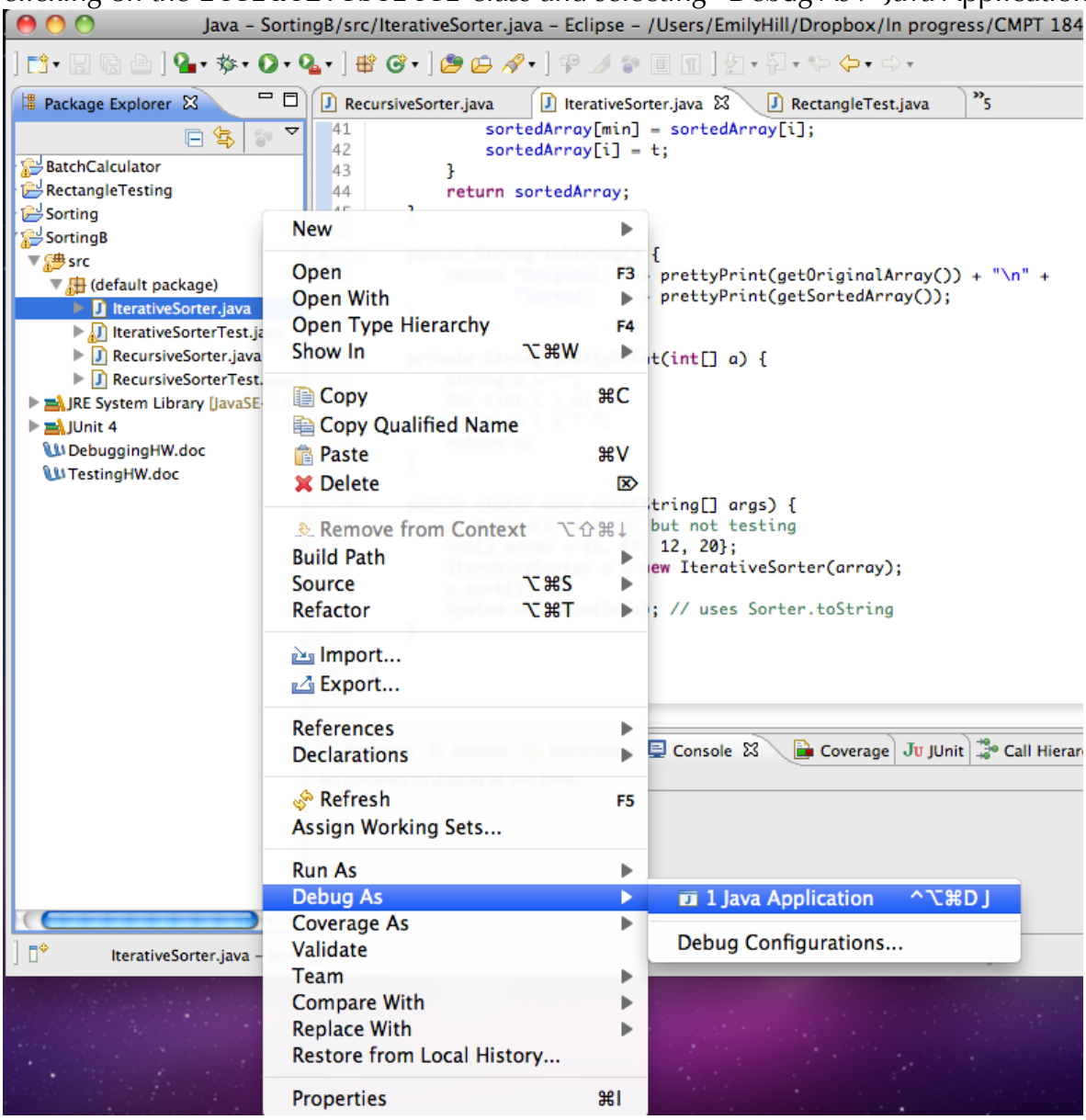
2. In the gray margin to the left of the line numbers, right click next to line 63 and select "Toggle Breakpoint":



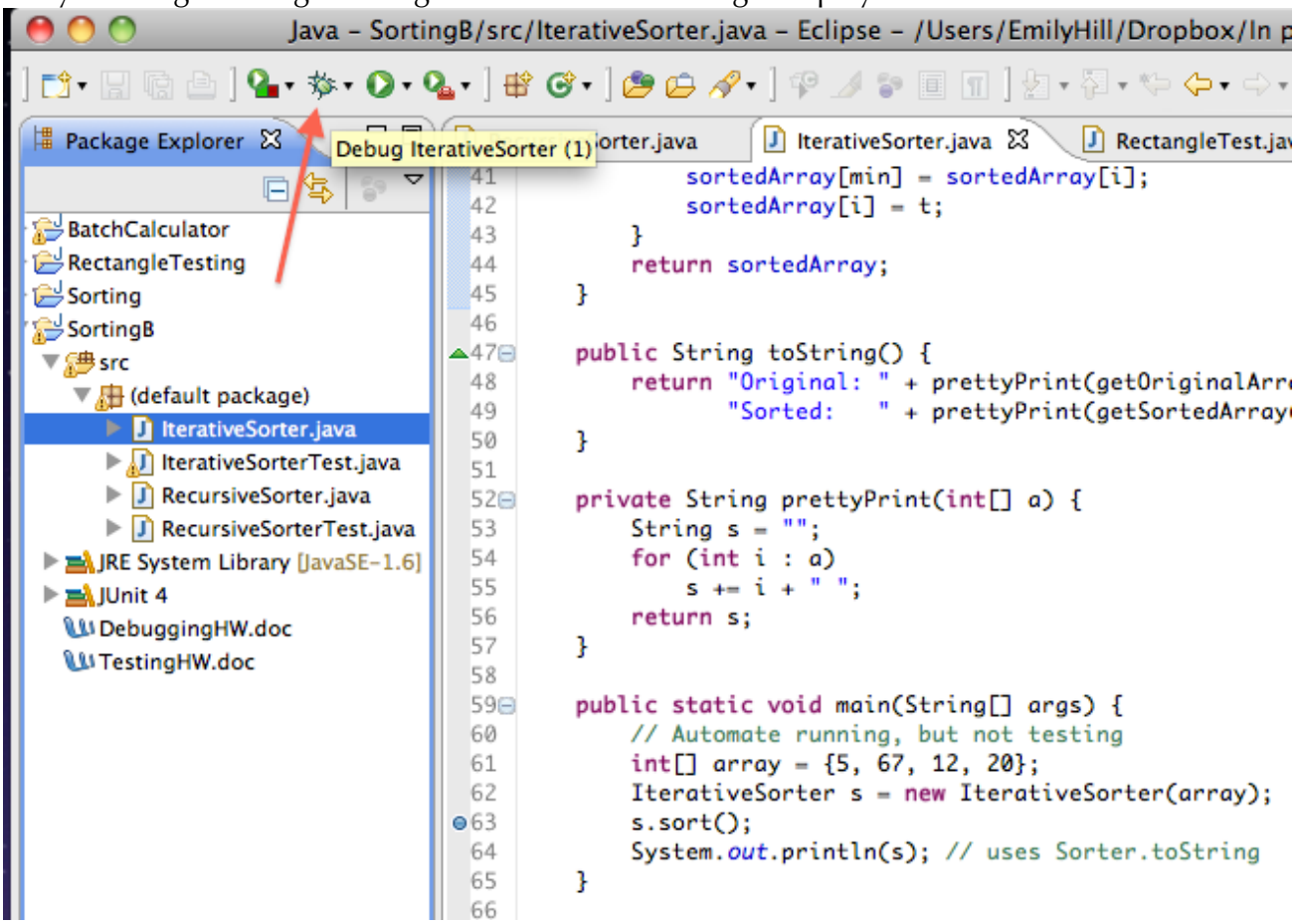
When the breakpoint is set, you should see a blue circle next to line 63:

```
59    public static void main(String[] args) {
60        // Automate running, but not testing
61        int[] array = {5, 67, 12, 20};
62        IterativeSorter s = new IterativeSorter(array);
63        s.sort();
64        System.out.println(s); // uses Sorter.toString
65    }
66
```

3. Now that we've set a breakpoint, we can begin debugging. We can do this by right-clicking on the `IterativeSorter` class and selecting "Debug As > Java Application"



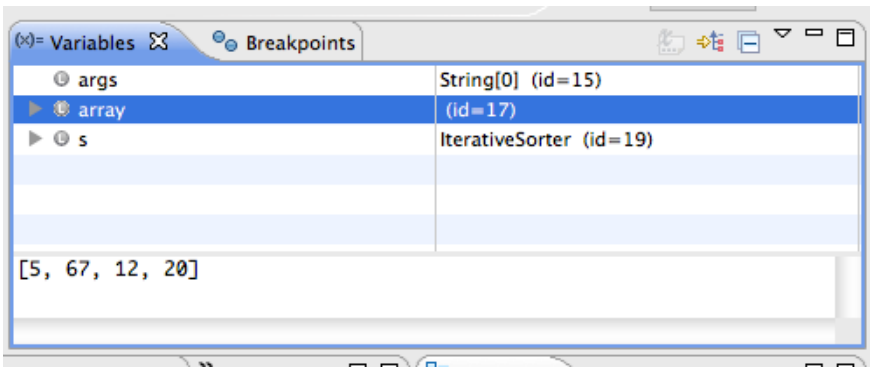
or by clicking on the green bug icon to the left of the green play arrow



4. Eclipse will ask if you'd like to open the debugging perspective, select "Yes" (I recommend selecting the check box so this becomes the default decision.

5. The debugging view has a number of panes:

Pane 1: in the top left we have the debugging view that shows us the call stack and gives us buttons to control the execution
Pane 2: in the top right we can control what breakpoints are set, and by clicking on "Variables", see what values our variables have during execution



For example, we see that our array contains the values [5, 67, 12, 20] -- just as we set them in line 61

Pane 3: what line we are in the process of executing is highlighted in the source code window. We can also edit our code on the fly (such as to fix bugs), and Eclipse will do its best to continue execution from the beginning of the current method call.
Panes 4 & 5 are the same console and outline views as we have in the java perspective, where we edited and tested code previously

6. Let's focus for a moment on the buttons in the debugging view:

-  Resume: allows us to resume execution until the next breakpoint is reached

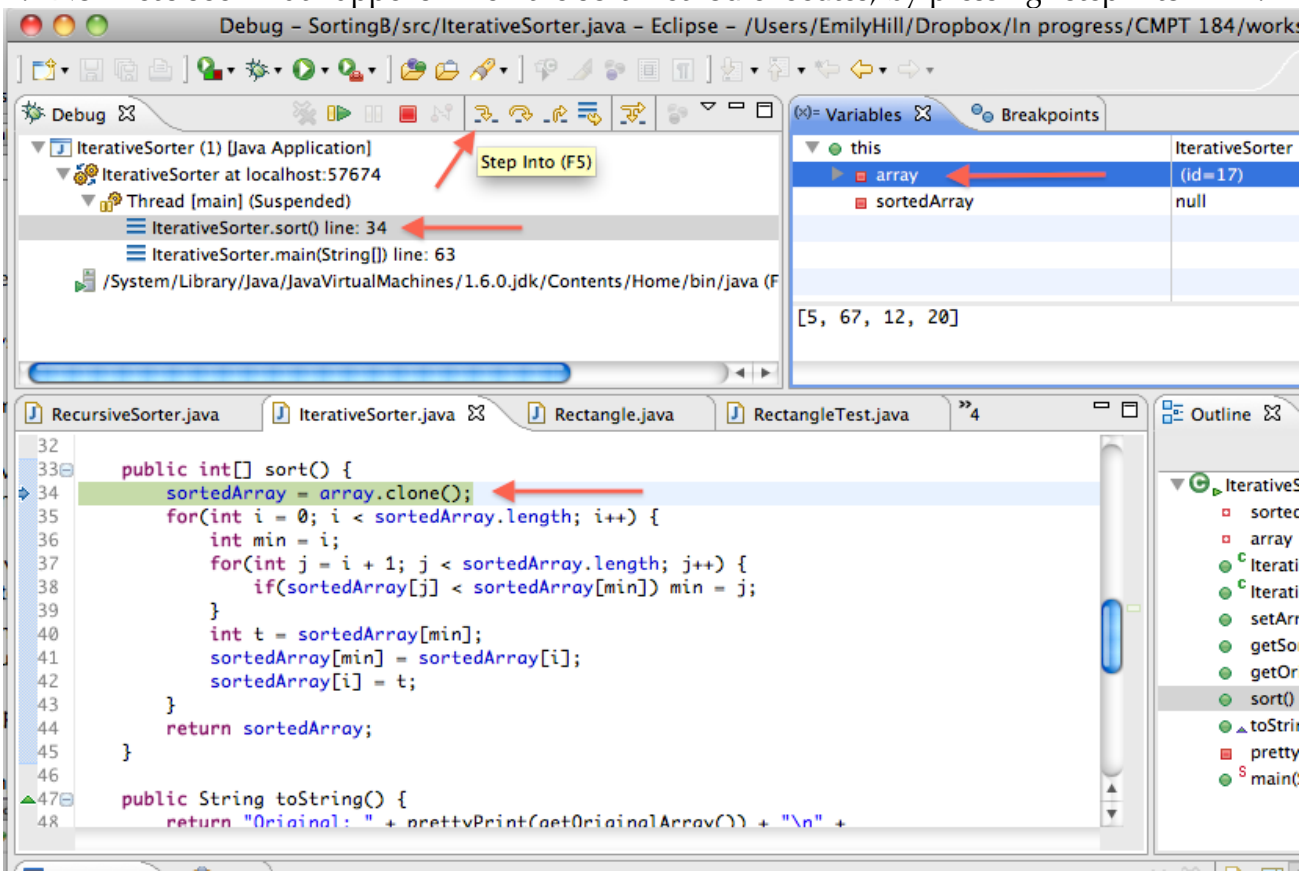
-  Terminate: immediately halts execution (same function as in the Java perspective)

-  Step Into: allows us to step into the execution of a method, so we can watch which lines are executed and how the values of variables change

-  Step Over: moves on to the next statement in the method we're executing; if the method is completed, step over behaves the same as step return

-  Step Return: when inside method calls other than main, allows us to finish executing the current method and return back to the caller

7. Now let's see what happens when the sort method executes, by pressing "step into"  :



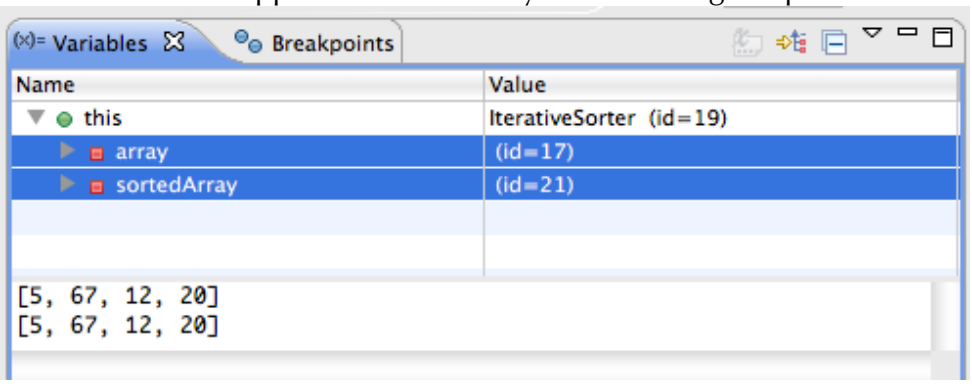
We see a number of changes:

- Our editor has changed to the sort method, and the first line is highlighted, ready to be executed next

- We have added a call to sort in our call stack

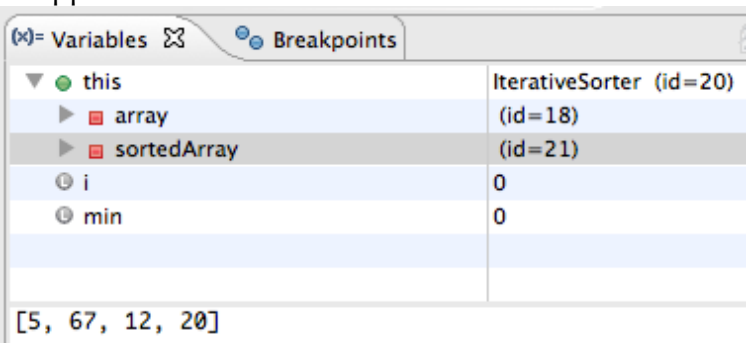
- By expanding "this" (our IterativeSorter object 's'), we can see the current values of the array and sorted array fields

8. Watch what happens to sortedArray after clicking “step over”

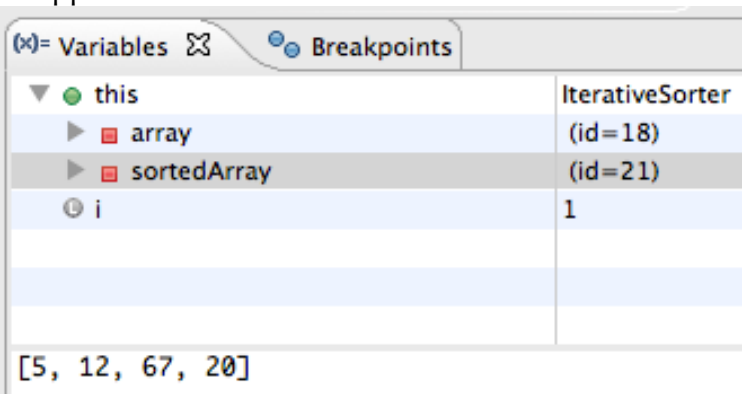


We can see that sortedArray and array both contain the same list of 4 numbers.

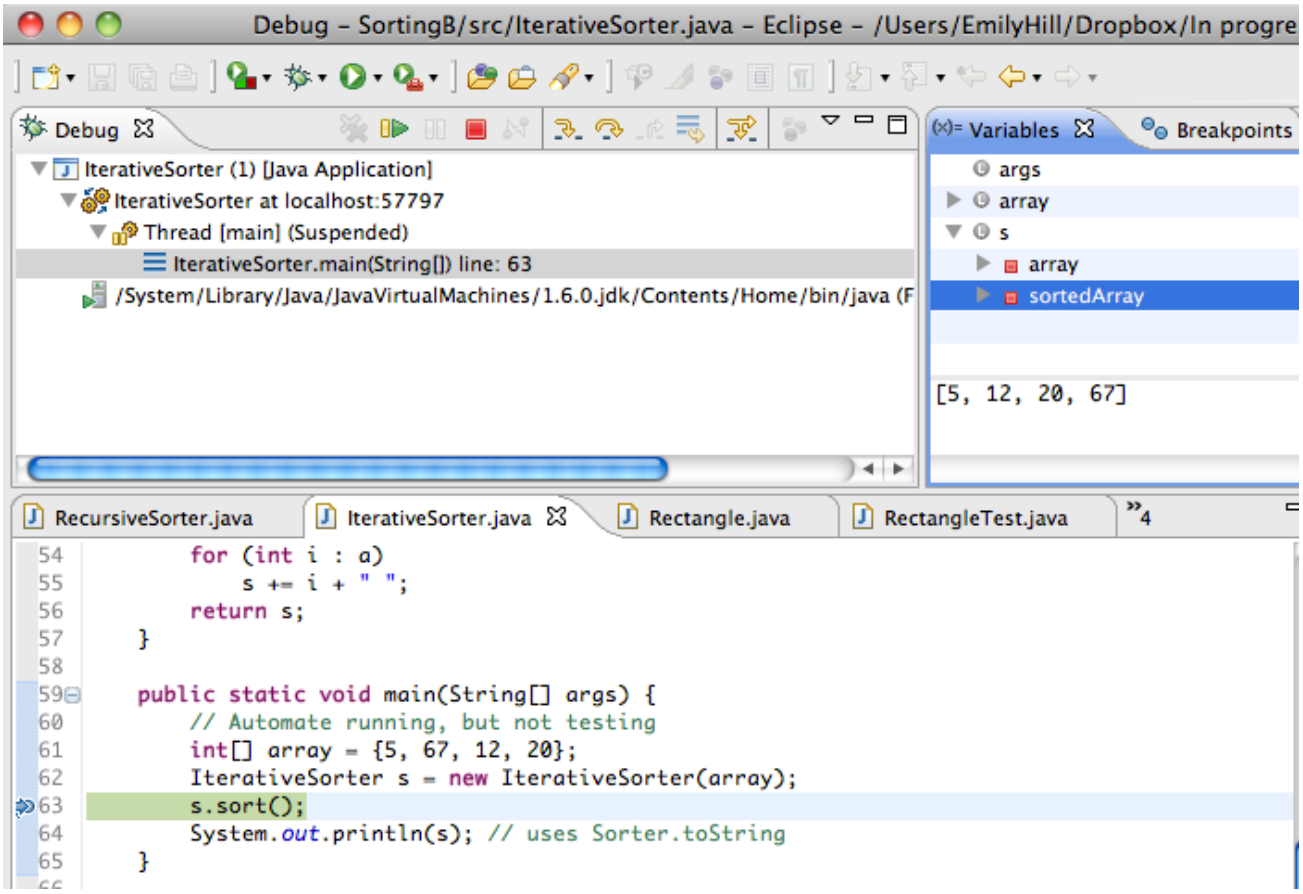
9. Line 35 begins a loop that finds the index of the minimum element in the array, and then in lines 40-42 swaps the smallest number with the minimum unsorted value in the list. Looking at sortedArray, we can predict that the index of the minimum element is “0”. Hit step over until you reach line 40. Is min set to “0” as we predicted? Which two numbers will be swapped in lines 40-42?



10. Continue stepping over lines until you reach the beginning of the loop at line 35. Try to predict what will happen in this iteration of the loop. Looking at the 3 elements starting at index 1, what is the index of the next minimum element? Which two numbers will be swapped? Step over the lines, checking the value of min at 40 and verifying sortedArray when you reach line 35 again to see if you correctly predicted which two numbers would be swapped.



11. Continue stepping through the execution until you understand how the algorithm for IterativeSorter works. When you’ve seen enough, press “step return” to go back to main



where we see that the sortedArray field in IterativeSorter variable 's' does indeed contain a sorted array of numbers

12. To complete your debugging session, click "resume" to execute line 64 and finish execution, or click "stop" to halt the execution.

13. To get back into editing mode, select the "Java" perspective in the top right