**Module 2 Lab – Bug Hunt**

Tim Mastarone

Rasmussen University

Advanced Java Programming

Instructor: Robert Kumar

Module 02-Lab Assignment

August 11, 2024

Java IDE Open:

A screenshot of a computer

Description automatically generated

**Using the Debugger:**

Program suspended while running with the debugger attached:  
A screenshot of a computer

Description automatically generated

**Breakpoint set**:

A screenshot of a computer

Description automatically generated

**3 Breakpoints set:**

A screenshot of a computer

Description automatically generated

The first breakpoint on line 4 pauses the program before anything is printed to the terminal. This allows for the user to step to the next breakpoint before any loops run. The second breakpoint pauses the program before each iteration of the outer loop (the ‘for’ loop that calls ‘innerloop()’). The third breakpoint is to pause the program after both loops have exited.

Below is a screenshot with the ‘innerloop()’ breakpoint removed. This allows the program to run once the first breakpoint is stepped over. It will run until the outer loop incrementor reaches 999,999 at this point the final breakpoint is hit.

A screenshot of a computer

Description automatically generated

**Stepping Control**

I added a breakpoint on line 16 when the ‘delay()’ method is called. I wanted to see how this loop executes so I used the ‘Step Into’ button. What this does is step you into the method code to see how it executes. If I use the ‘Step Over’ button it executes the method call but the debugger does not enter the method, just calls the method and stops at the next point.  
Once I ‘stepped into’ the ‘Delay()’ method I used the ‘Step Return’ button to get out of the inner code and return to the main execution.

A screenshot of a computer

Description automatically generated

**Evaluating Expressions:**

Watch set for the expression ‘J < 100000’:  
A screenshot of a computer

Description automatically generated

**Fixing Buggy String Methods using the Eclipse Debugger**

Issue #1: countOccurences returns a value 1 higher than the actual characters matched. If no characters are matched it returns a 1. I set a breakpoint and created variable and expression watches to find this:

A screenshot of a computer

Description automatically generated

I then stepped into the countOccurences() method and found the issue. The Count variable is initialized to ‘1’ instead of ‘0’ causing this issue. Another problem is the for loop starts at ‘1’ as well skipping over the first character.

Stepped into the countOccurences method:

A screenshot of a computer

Description automatically generated

The snip below shows the corrected code:

A screenshot of a computer

Description automatically generated

Now the characters are accurately counted and the proper result is displayed to the user.

Issue #2:

The reverse string function throws an exception and shuts down. There is an array bounds issue so I set a breakpoint in the reverseString() method. When I got there I saw the for loop was also starting at ‘1’. I changed this to a ‘0’ along with making the exit condition ‘<’ rather than ‘<=’ to make this work.

A screenshot of a computer

Description automatically generated

This got me further, but another issue was apparent when I set a watch on ‘swap.Length – i’. This showed me that the ‘for’ loop trying to access an array index 1 higher than the bounds.

By changing this to ‘(swap.Length – 1) – i’ the loop operates correctly. The corrected code is below:

A screenshot of a computer

Description automatically generated

As shown above, this allowed the program to execute but presented another issue.

Issue #3:

The ‘Reversed’ string is the same as the string entered.

I set watches for expression and variables in this for loop to find that the ‘for’ loop iterates over the entire string. I found after stepping through half of the iterations the string was reversed. This indicates we only need to go over half the length of the string (char array) in the ‘for’ loop. The snip below show the watch list in the debug view:

A screenshot of a computer

Description automatically generated

The code below is corrected to only go over the half of the string when reversing characters:

A screenshot of a computer

Description automatically generated

The snip below shows the code is functioning properly now:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Fixing Guess2.Java:**

**Issue #1**: When running the program the first issue is correct guesses do not increase user points. Whether the guess is correct or not the user is prompted for another guess. I set a breakpoint after the do while loop to see what was happening:

A screenshot of a computer

Description automatically generated

With watches on ‘rolled’ and ‘userguess’ I saw there was a match, and stepped into the next block of code. Since the do while loop is looking at the boolean ‘rightGuess’ the loop ran again. I see that the exit condition ‘rightGuess’ is never being set to ‘true’ causing the infinite loop. I added this to the ‘if’ block when ‘rolled’ = ‘userguess’:

A screenshot of a computer

Description automatically generated

**Issue #2**: Invalid guesses do not display a different user message. I used the breakpoint mentioned above in the do while loop to find the condition was never being met. This is due to the compound condition using an ‘and’ rather than ‘or’.

The corrected code is below, and the condition is met when invalid characters are entered by the user:  
A screenshot of a computer

Description automatically generated

**Issue #3**: More than 3 guesses are allowed if they are not correct. I set a breakpoint on the while condition and realized that the ‘numGuesses’ is over 3, but since the guesses were incorrect the ‘or’ case allowed the while block to be entered again. I changed this to an ‘and’ so that the guess count must be below 3 and the rightGuess must be false. If either condition is not met this loop should not be entered. The snip below shows the modified code working:

A screenshot of a computer

Description automatically generated

After 3 rounds of incorrect guesses the program moves on to round 2 of 5.

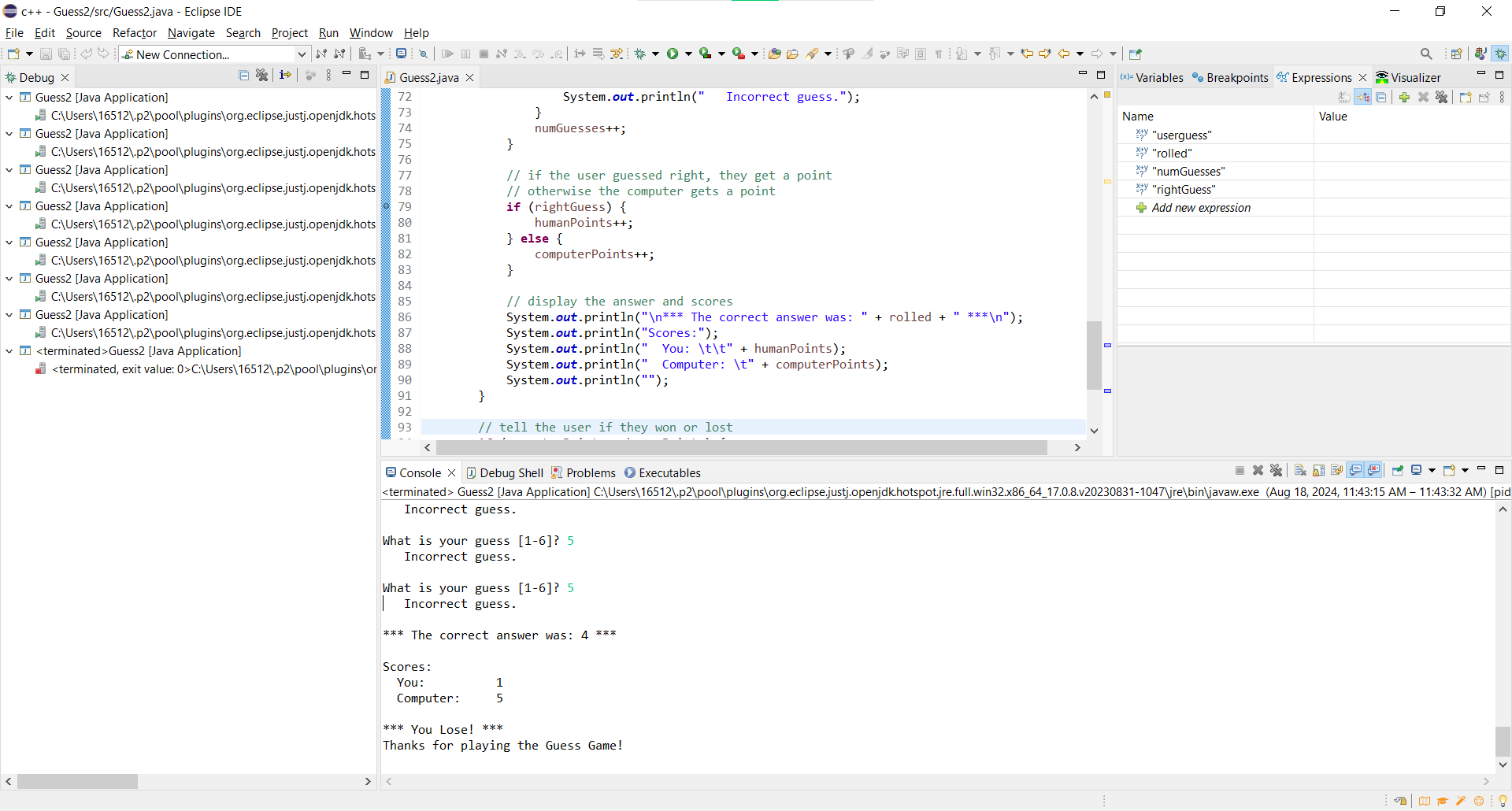
**Issue #4**: Now the program executes further, but the ‘while’ loop is not executed during the second round. Using the same breakpoint on the while condition, I was able to see the ‘numGuesses’ was still set to 3. This kept the while loop from executing. This should be reset to ‘0’ for the second round like the code below:

A screenshot of a computer

Description automatically generated

This allowed the full program to run. There is still one issue with point tracking.

**Issue #5**: Player receives a point on incorrect guess, and computer receives a point when the guess matches. This is inverted behavior in comparison with the program instructions. TO figure this out I removed existing breakpoints and placed one on line 79 when the points are assigned using an ‘if-else’ block. Stepping over I could see the point assignment was inverted. This can be easily fixed in 2 ways, using the ‘!’ not operator in front of ‘rightGuess’, or swapping the code blocks. I chose the ladder as it is more intuitive for the reader. The snip below shows the modified code that ran until completion of the program:



The program now properly runs 5 rounds, and the score is kept correct.

**BuggyQuilt.Java**:

When attempting to run BuggyQuilt.Java I see an exception on array bounds right away:

A screenshot of a computer

Description automatically generated

The stack trace indicates an error in the ‘createFlipped()’ method so I placed a breakpoint there. When running with the debugger attached I see the code running before the exception. I stepped through the block of code and added watches to the arrays and variables:

A screenshot of a computer

Description automatically generated

I added more watch statements to see that the row index being referenced was indexed to the length rather than the index of ‘col’ causing the out of bounds issue. I subtracted 1 from ‘blockRows’ before assigning it to ‘int flippedRow’. This gets the program further.

Another bounds issue come up when I run the program now, referring to the method ‘placeBlock’. I moved the breakpoint to line 42 to analyze the code:  
A screenshot of a computer

Description automatically generated

I could see here the arguments are sent in the wrong order in the second instance of ‘placeBlock()’. This causes an array index out of the range to be passed.

After fixing this there are still errors. When breaking in the ‘createFlipped()’ method I see the blockRows and blockCols are set to the same value (rows length). The column value should be assigned the length of ‘block[0]’ to get the accurate column count.

When stepping through I watched the ‘flippedRow’ variable to find it was not being decremented in the loop. I added these corrections in the code below:  
A screenshot of a computer

Description automatically generated

The last change I had to make was fixing the arguments sent to ‘placeBlock’ on line 30. The ‘c’ and ‘r’ counts were being assigned out of order. I copied the assignment from the working ‘placeBlock’ call and it fixed the issue.

A screenshot of a computer

Description automatically generated

After these changes the program runs to completion as shown below:  
A screenshot of a computer

Description automatically generated