# 1. Getting Familiar with Debugging in Eclipse

**SCREEN CAPTURE ONE**

|  |
| --- |
| **VARIABLES WINDOW:**  Graphical user interface, application  Description automatically generated |

TABLE ONE

|  |
| --- |
| **Question: What is the Eclipse keyboard shortcut for toggling a breakpoint** |
| **Answer:**  CTRL + Shift + B (Windows)  CMD + Shift + B (MacOS) |
| **Question What is the difference between “Step-Over”, “Step-Into”, and “Step-Return”?** |
| **Answer:**   * Step-Over – Execute the function and you can see inside the function and see how the function is executing line by line. Then returns you to the next line right after the function call. * Step-Into – Execute the function call and returns the result but you cannot see how the function was executed. * Step-Return – Execute until the end of the current function, stop after calling line. |
| **Task: Practice tracing through the DebugStar sample program.**  **It is ok if you don’t understand all the java code; but you should be able to trace the order in which statements are executed.**  **Based on your best understanding of the program, provide a list of methods that are called when the program executes (from start to end, in order of being called). You can skip library methods (like println, for example).  HINT: Use a combination of “Step-Into” “Step-Over” and “Step-Return”. Use the “Stack Trace” window.** |
| **List of Methods (in order below). Please use the fully qualified name, eg. “Debugstar.run(String, int, int)”. Use the stack view to help you.**  **Answer:**   1. DebugStar.main(String[]) line:8 2. DebugStar.run(int) line: 33 3. Factorial.perform(int) line: 62 4. Factorial.perform(int) line: 61 5. Factorial.perform(int) line: 64 6. DebugStar.run(int) line: 33 7. DebugStar.run(int) line 34 8. DebugStar.run(int) line: 35 9. DebugStar.main(String[]) line: 9 |

# 2. The Debug Challenge

**You are given a source code for a program that is “BUGGY” called FibonacciBuggy. Your job is to find out why, using the skills that you have acquired so far.**

1. SCREEN CAPTURE: Original Code with line number

|  |
| --- |
| Text  Description automatically generated |

2. Provide an error log table indicating error details (line number, type of error, and explain error and show correction).

|  |  |  |  |
| --- | --- | --- | --- |
| Line(s) | Type of error (compile-time, run-time, or logical) | Description | Correction |
| 28 | Compile-time error | The local variable declaration statement needs to end with a semicolon. | int n; |
| 31 - 34 | Compile-time | Changed the while loop into a do-while loop | do {  } while |
| 39 | Compile-time error | Closed the scanner | scanner.close() |
| 49 | Run-time error | Changed the second get method to (i – 2), so the while loop matches the Fibonacci formula | (i – 2) |
| 60 | Logical error | Changed get(i) into get(i – 1) | Get(i – 1) |
| 61 |  | Added empty space in between numbers using System.out.print(“ “); | System.out.print(“ “); |

3. Provide screen capture of fixed code (with line numbers), and sample run using n=10:

|  |
| --- |
| Text  Description automatically generated |