Java Basics – Debugging

The goal of this lab is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to pinpoint the bug(s) and fix it (without rewriting the entire code).

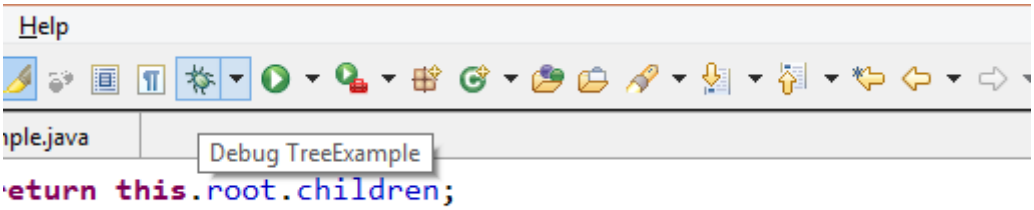
Problem 1. Set up your debugger in Eclipse

Your first task is to configure the **debugger** in Eclipse so that you are comfortable using it.

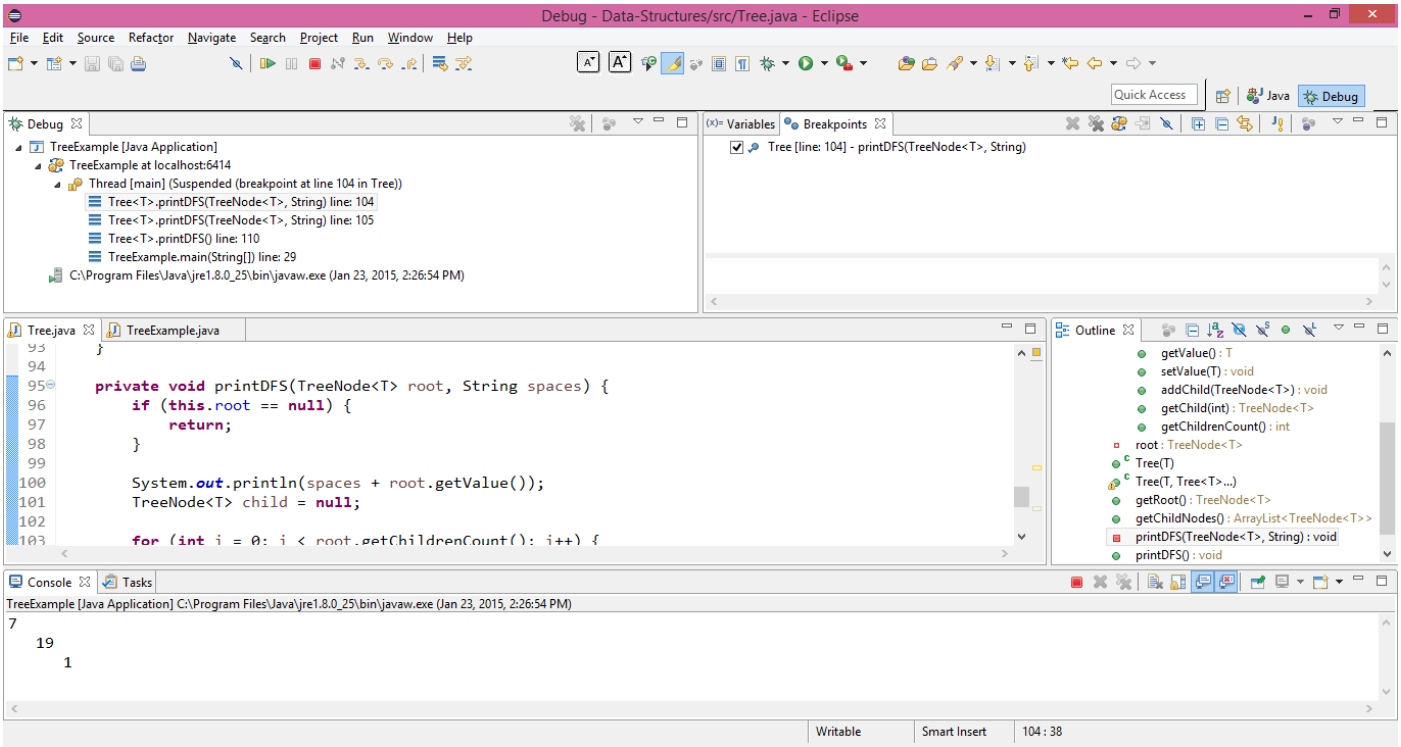
Set a breakpoint in your program:

```
103         for (int i = 0; i < root.getChildrenCount(); i++) {
104             child = root.getChild(i);
105             printDFS(child, spaces + "    ");
106         }
107     }
```

And start the debugger:



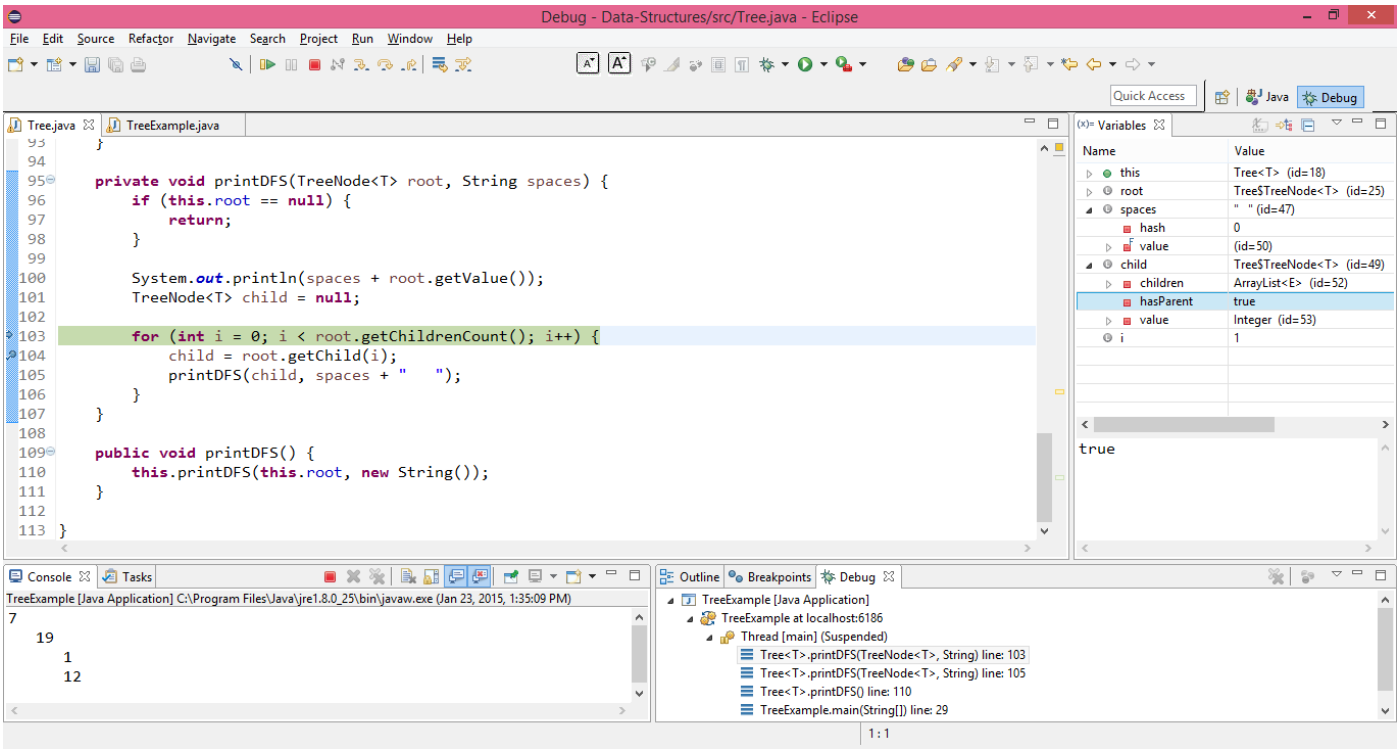
The debugging perspective should appear:



However, it's pretty messy. Modify it to your taste by moving the windows around. The following windows should be visible:

- **Variables** – shows all local variables and their value.
- **Debug** – shows the **call stack** (all the methods in the order they were called + line number).
- **Breakpoints** – displays all the breakpoints placed.
- **Console** – shows the output in the console.

The result should be something much more structured.



Problem 2. Placing Breakpoints

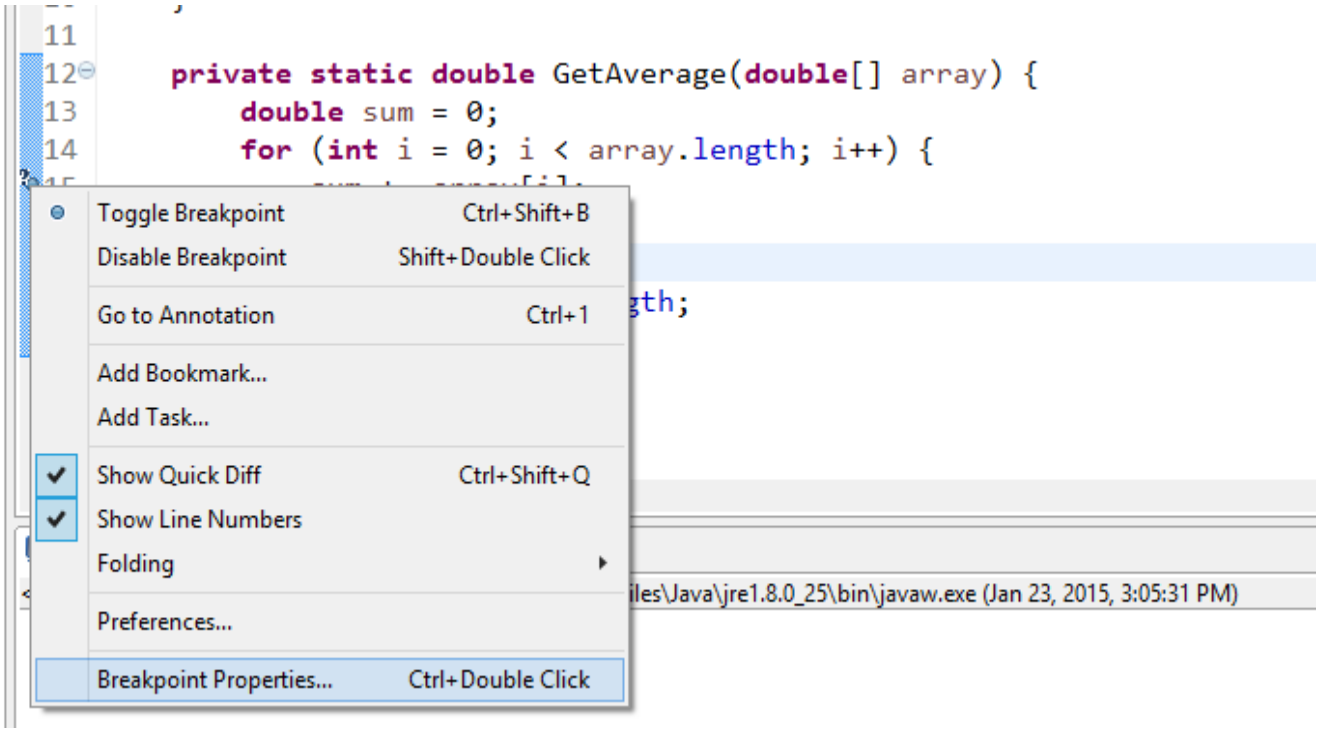
Open the provided **AverageSum.java** program for practicing debugging.

- A **breakpoint** stops execution at a specific line of code. To add a breakpoint, click to left of a line number.

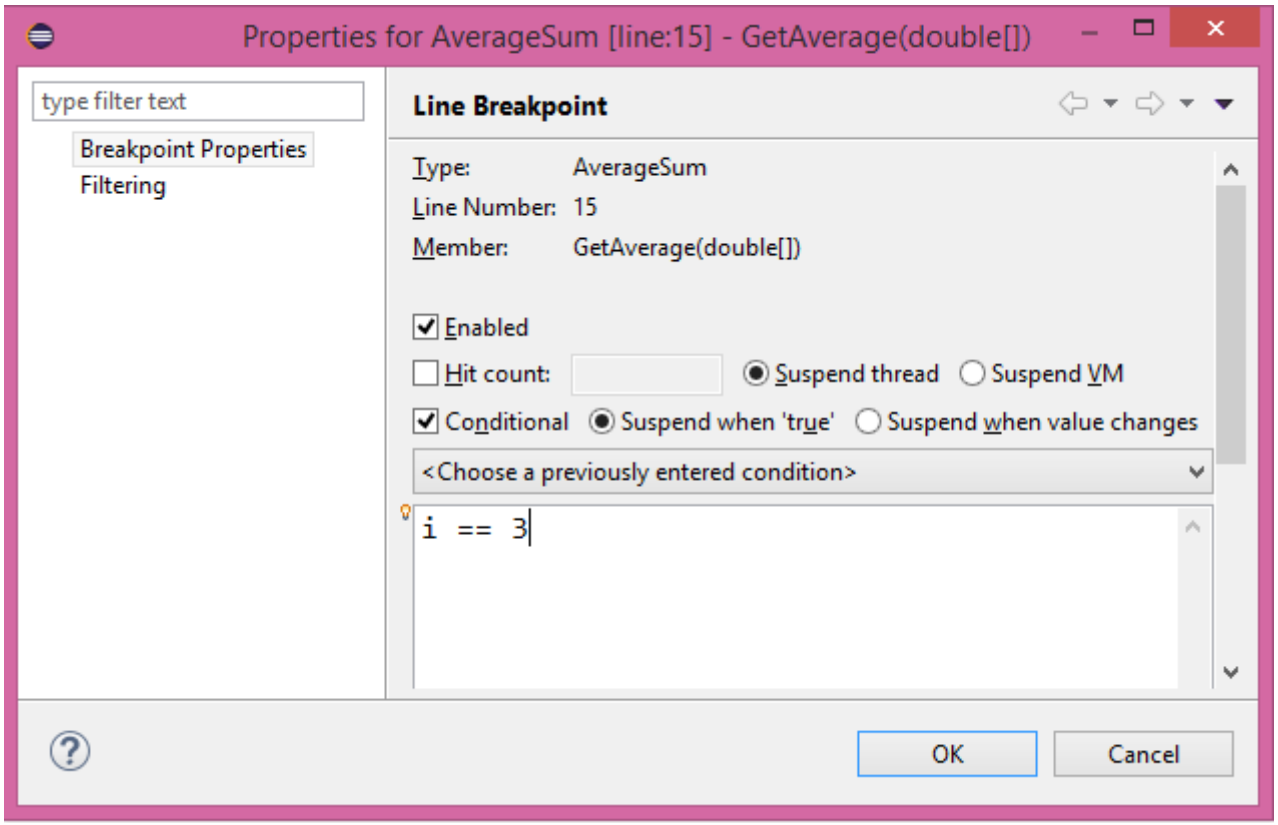
```
103         for (int i = 0; i < root.getChildrenCount(); i++) {
104             child = root.getChild(i);
105             printDFS(child, spaces + "    ");
106         }
107     }
```

  The next time the program is run in **debug mode**, it will stop execution when/if it reaches that line of code, allowing us to analyze the program variables.

- A **conditional breakpoint** stops execution only if a certain condition is met (e.g. `i == 3` in a loop). It is added by setting a normal breakpoint > **right click** > **Breakpoint Properties**.



We then a condition (e.g. **`i == 3`**).



When we debug, the program will stop execution at the given line **only** when **i = 3**.

```
11
12      private static double GetAverage(double[] array) {
13          double sum = 0;
14          for (int i = 0; i < array.length; i++) {
15              sum += array[i];
16          }
17
18          return sum / array
19      }
20  }
21
22
```

i= 3

3

Problem 3. Stepping through the code

When debugging, we can go line by line and analyze how our variables change as our program executes.

The **current line of execution** is shown by a blue arrow called the Current Instruction Pointer.

```
12      private static double GetAverage(doub
13          double sum = 0;
14          for (int i = 0; i < array.length;
15              sum += array[i];
16          }
17
18          return sum / array.length;
19      }
20  }
21
```

Stepping through our lines of code can be done using the following commands:

- **Step into (F5)** – executes the current line and enters the method
- **Step over (F6)** – executes the current line without entering the method
- **Step return (F7)** – exits the current method
- **Resume (F8)** – resumes program execution to the end (stops if it reaches another breakpoint)

Or by using the toolbar on the top: