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:

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

And start the debugger:

```
Help

I to the light of the li
```

The debugging perspective should appear:

```
File Edit Source Refactor Navigate Search Project Run Window Help
X 🗩 II 🔳 18 3. 🖘 R 📑 📆
                                                                      Quick Access 😭 🐉 Java 🏌 Debug
                                                                     🗽 🍃 🔻 🗆 🗎 🗀 (x)= Variables 💁 Breakpoints 🛭
                                                                                                                                  🏇 Debug 🏻
▲ J TreeExample [Java Application]

✓ > Tree [line: 104] - printDFS(TreeNode<T>, String)
   ■ Tree<T>.printDFS(TreeNode<T>, String) line: 104
■ Tree<T>.printDFS(TreeNode<T>, String) line: 105
          Tree<T>.printDES() line: 110
     C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (Jan 23, 2015, 2:26:54 PM)
                                                                                                                          🚺 Tree.java 🛭 🔝 TreeExample.java
                                                                                                                                              getValue() : T
  94
                                                                                                                                           setValue(T) : void
  95⊜
          private void printDFS(TreeNode<T> root, String spaces) {
                                                                                                                                             addChild(TreeNode<T>): void
 96
97
               if (this.root == null) {
                                                                                                                                           getChild(int) : TreeNode<T>
                                                                                                                                           getChildrenCount(): int
root: TreeNode<T>
                   return;
  99
                                                                                                                                           Tree(T)
 100
               System.out.println(spaces + root.getValue());
                                                                                                                            П
 101
               TreeNode<T> child = null;
                                                                                                                                        getRoot(): TreeNode<T>
                                                                                                                                           getChildNodes() : ArrayList<TreeNode<T>>
 102
               for (int i = 0: i < root.getChildrenCount(): i++) {</pre>
                                                                                                                                          printDFS(TreeNode<T>, String): void
                                                                                                                                   □ Console 🏻 🔎 Tasks
TreeExample [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (Jan 23, 2015, 2:26:54 PM)
   19
                                                                                                  Writable
                                                                                                              Smart Insert 104 : 38
```















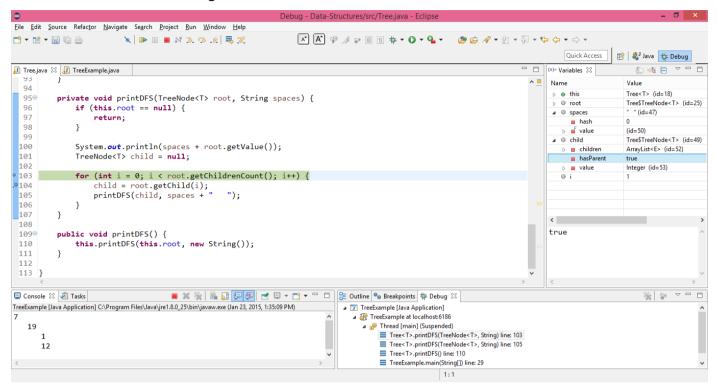




However, it's pretty messy. Modify it to your taste by moving the windows around. The following windows should be visible:

- 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.

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

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**.



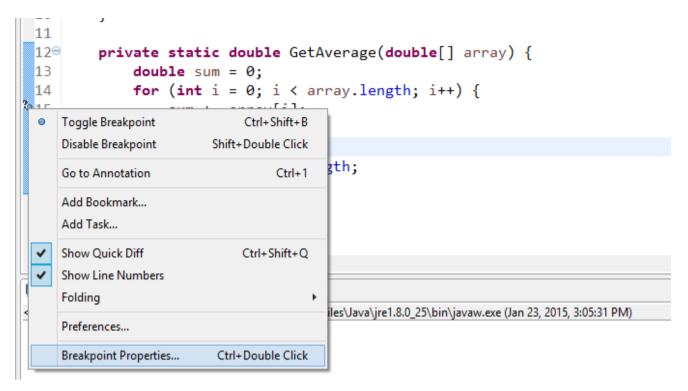




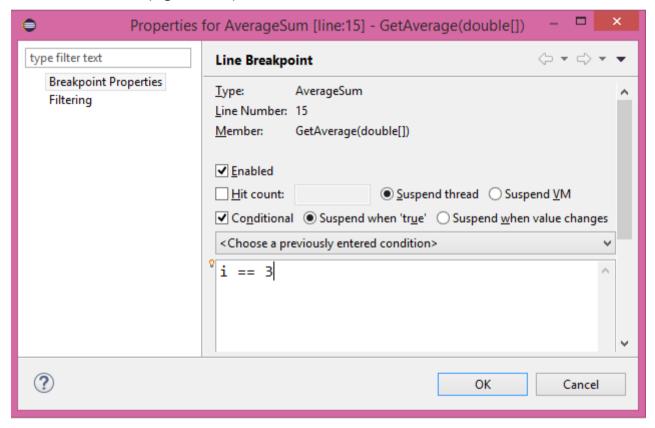








We then a condition (e.g. i == 3).



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;
             for (int i = 0; i < array.length; i++) {</pre>
 14
15
                  sum += array[i];
 16
                                     0 i= 3
 17
 18
             return sum / array
 19
         }
 20
 21
 22
```

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;</pre>
215
                  sum += array[i];
 16
 17
             return sum / array.length;
18
 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:



















