for(int i=0;i<5;i++){ //line1

System.out.println("gvtrf"); //line2

} //line3

System.out.println("frefgvtrf"); //line4

* Normally we can define a formula like N(S+C)\*iterations to calculate a complexity of a simple code.
* There N is nesting level calculated after indentation happened; S is size which calculates the count line wise basically a program statement. We can ignore operators like (), {}, ; and etc as they are depending upon the programming language even. And in the above formula “C” is the control structures like for, if and etc where weights can be given 3,2 and etc respectively because in “**If”** there is only to check one or more conditions but in a loop like “**for”** there may have condition(s) plus the initializations and increments. And also we should multiply the complexity inside the loop by number of iterations. But we should skip multiplication of variable initialization and condition wisely. (remember about the for loop execution in DAA)
* In the above code I have counted ++ as two symbols because it’s kinda summation, where Halstead is wrong on this.
* I considered the count of “System.out.println” as one because it’s like one program keyword to print the output.

When the loop condition gets **true**,

|  |  |  |  |
| --- | --- | --- | --- |
| line | Size(S) | Control Structures(C) | Complexity N(S+C)\*iterations |
| 1 | 10 | 3 | 1(10+3)\*1 = 13 |
| 2 | 2 | 0 | 2(2+0)\*5 = 20 |
| 3 | 0 | 0 | 0 |
| 4 | 2 | 0 | 1(2+0)\*1 = 2 |

Total complexity when it’s true = 13+20+0+2=35

When the loop condition gets **false**,

|  |  |  |  |
| --- | --- | --- | --- |
| line | Size(S) | Control Structures(C) | Complexity N(S+C)\*iterations |
| 1 | 7 | 3 | 1(7+3)\*1 = 10 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 2 | 0 | 1(2+0)\*1 = 2 |

Total complexity when it’s false = 10+0+0+2=12

By considering both true and false total complexity can be derived using the **Median** of both true and false

**Total complexity**= (35+12)/2 = 23.5