package com.setm.metric; //line 1

//line 2

import org. slf4j.Logger; //line 3

import java.net.UnknownHostException; //line 4

//line 5

public class NewMetric { //line 6

private final Environment env; //line 7

public static void main(String[] args) throws UnknownHostException { //line 8

Environment env = app.run(args).getEnvironment(); //line 9

String protocol = "http"; //line 10

if (env.getProperty("server.ssl.key-store") != null) { //line 11

protocol = "https"; //line 12

} //line 13

} //line 14

} //line 15

* Normally we can define a formula like N(S+U+C)\*iterations to calculate a complexity of a simple code.
* There N is nesting level, S is size which calculates the count line wise basically a program statement without counting coding standards. 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 plus the initializations and increments. And also we should multiply the complexity inside the loop by number of iterations.
* I have counted com.setm.metric in line 1 as one symbol because it is actually one package name.
* I have counted import org. slf4j.Logger in line 3 as an unused import where we can count under U. There that calculation is also same as calculating S but need to emphasize we are considering the factor of coding standard separately I counted it under U.
* In line 4 it’s a used import and counted “import”, “java.net.UnknownHostException” as two separate symbols.
* I have considered “public”, “class” , “NewMetric” as three symbols in line 6 and get counted even it’s a class declaration because they are kinda program statement.
* Even though Halstead doesn’t consider env as an operator or operand, I’ll count as one symbol in line 7.
* I have considered “main” as a symbol and get counted even it’s a method declaration because they are kinda program statement. And also counted args in line 8 too by taking as a program segment.
* In line 9, app.run(args) => app.run() consider as one symbol because no point of counting app and run() separately since run() is belonging to app. And as usual argument “args” taken as one symbol. Then .getEnvironment() is counted separately even it also belongs to app.run(args) but to get the output of .getEnvironment(), we have to call app.run(args) first. So that’s the reason for the separation of app.run(args) and .getEnvironment(). (Remember we don’t count . separately)

When line **11** condition gets **true** complexity should be as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| line | S | U | C | N(S+U+C)\*iterations |
| 1 | 2 | 0 | 0 | 1(2+0+0)\*1 = 2 |
| 3 | 0 | 2 | 0 | 1(0+2+0)\*1 = 2 |
| 4 | 2 | 0 | 0 | 1(2+0+0)\*1 = 2 |
| 6 | 3 | 0 | 0 | 1(3+0+0)\*1 = 3 |
| 7 | 4 | 0 | 0 | 2(4+0+0)\*1 = 8 |
| 8 | 9 | 0 | 0 | 2(9+0+0)\*1 = 18 |
| 9 | 6 | 0 | 0 | 3(6+0+0)\*1 = 18 |
| 10 | 4 | 0 | 0 | 3(4+0+0)\*1 = 12 |
| 11 | 5 | 0 | 2 | 3(5+0+2)\*1 = 21 |
| 12 | 3 | 0 | 0 | 4(3+0+0)\*1 = 12 |
| 13-15 | 0 | 0 | 0 | 0 |

Total complexity when 11 condition gets true = 2+2+2+3+8+18+18+12+21+12+0 = 98

When line **11** condition gets **false** complexity should be as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| line | S | U | C | N(S+U+C)\*iterations |
| 1-11 | Same as above | Same as above | Same as above | Same as above |
| 12-15 | 0 | 0 | 0 | 0 |

Total complexity when 11 condition gets true = 86+0 = 86

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

**Total complexity**= (98+86)/2 = 92