Project Specification

The Story

ABC Solutions is a startup and is evolving rapidly. At present they only concentrate on developing C++ and Java based projects. The primary objective of ABC solutions is to build quality software that could be easily maintained. Assume your group is working for ABC solutions. In an attempt to reduce the maintenance cost of the software developed by the company, the CEO has requested your group to **develop a complexity measuring tool**. Accordingly, he wants to measure the complexity introduced due to the following factors:

a. Size
b. Type and the nesting level of control structures
c. Inheritance
d. Recursion

Measuring the complexity of a program statement due to size (Cs)

- A constant value of **two** is added to **'Cs'** at the detection of the following:
  - Reference (&) and dereference (*) operators
  - 'new', 'delete', 'throw', and 'throws' key words

Note: The (*) sign used in the declaration of C++ pointer is not a **dereference** operator. It is just a similar notation that creates a pointer.

- A constant value of **one** is added to **'Cs'** at the detection of the following:
  - Arithmetic operators → { + - * / % ++ -- }
  - Relation operators → { == != > < >= <= }
  - Logical operators → { && || ! }
  - Bitwise operators → { | ^ ~ << >> >>> <<< }
  - Miscellaneous operators → { , -> . :: }
  - Assignment operators → {+= -= *= /= = >>>= |= &= %= <<= >>= ^= }
  - Key words → {void, double, int, float, string, printf, println, cout, cin, 'if', 'for', 'while', 'do-while', 'switch', 'case' etc.}
  - Manipulators → {'endl', ' \n', etc. }
  - Text inside a pair of double quotes → {Eg: "The greatest is" }
  - Class, method, object, variable, and array names
  - Numeric values (numbers)

Note: Key words such as **public**, **static**, **else, try** and **return** are not considered under the **size complexity factor**.

Measuring the complexity of a program statement due to type of control structures (Ctc)

- For a program statement with a **conditional control structure** such as an **'if'** condition, a weight of **one** is assigned for the **'if'** condition and for **each logical** ('&&' and '||') or **bitwise** ('&' and '|') operator that is used to combine two or more conditions.
- For a program statement with an **iterative control structure** such as a **'for', 'while', or 'do-while'** loop, a weight of **two** is assigned for the **'for', 'while', or 'do-while'** loop and for **each logical** ('&&' and '||') or **bitwise** ('&' and '|') operator that is used to combine two or more conditions.
- A weight of **one** is assigned for a program statement with a **'catch'** statement.
- A weight of **n** is assigned for a program statement with a **'switch'** statement with **n** number of cases.
- A weight of **zero** is assigned for all the other program statements in a program.

Measuring the complexity of a program statement due to nesting of control structures (Cnc)

- A weight of **zero** is added for program statements which does not contain **any level of nesting**.
- A weight of **one** is added for program statements which are at the **outer most level of nesting**.
- A weight of **two** is added for program statements which are at the **next inner level of nesting**.
- Similarly, the weight allocated for the program statements is increased by **one** for **each level of nesting**.

Measuring the complexity of a program statement due to inheritance (Ci)

- The complexity of all the program statements which belongs to a class is assigned the same weight that the class has due to its inheritance:

Complexity of a program statement of a class due to inheritance (Ci) = Complexity of the class due to its inheritance (CCi)

- Complexity of a class due to inheritance (CCi) is computed as follows:

Complexity of a class due to its inheritance (CCi) = Number of ancestor classes of the class + 1

Measuring total complexity of a program statement

- First, compute **total weight (TW)** of a program statement as follows:

$$TW = Ctc + Cnc + Ci$$

- Next, compute the **complexity of a program statement (Cps)** as follows:

$$Cps = Cs * TW$$

Measuring the complexity introduced due to recursion (Cr)

- Double the **Cps** values derived for each program statement that belongs to a recursive method.

Measuring complexity of a program (Cp)

- The complexity of a program (Cp) which consists of a **recursive method** is computed as follows:

Cp = Addition of **'Cps'** values derived for the program statements which does not belong to a recursive method + Addition of **'Cr'** values derived for the program statements that belongs to a recursive method

- The complexity of a program (Cp) which does **not** consist of a **recursive method** is computed as follows:

Cp = Addition of the **'Cps'** values of all the program statements in a program

Displaying the complexity of a program (Cp) which consists of a recursive method

Line no	Program statements	Tokens identified under the size factor	Cs	Ctc	Cnc	Ci	TW	Cps	Cr
1	public class FibonacciMain {								
2	public static long fibonacci(long number) {	long, fibonacci, long, number	4	0	0	1	1	4	8
3	if ((number == 0) || (number == 1)) { // base cases	if, number, ==, 0, ||, number, ==, 1	8	2	1	1	4	32	64
4	return number;	number	1	0	1	1	2	2	4
	}								
5	else {								
	// recursion step								
6	return fibonacci(number - 1) + fibonacci(number - 2);	fibonacci, number, -, 1, +, fibonacci, number, -, 2	9	0	1	1	2	18	36
	}								
	}								
7	public static void main(String args[ ]) {	void, main, String, args	4	0	0	1	1	4	
8	for (int count = 0; count <= 10; count++){	for, int, count, =, 0, count, <=, 10, count, ++	10	2	1	1	4	40	
9	System.out.println("Fibonacci of " + count + " is " + fibonacci(count));	System, ·, out, ·, println, "Fibonacci of ", +, count, +, " is ", +, fibonacci, count	13	0	1	1	2	26	
	}								
	}								
	}								
		Cp							**182**

Note: Program statements which contain class declarations and program statements which appear before class declarations are not considered for the complexity calculation.

Displaying the complexity of a program (Cp) which does not consist of a recursive method

Line no	Program statements	Tokens identified under the size factor	Cs	Ctc	Cnc	Ci	TW	Cps	Cr
1	import java.io.*;								
2	public class MyException {								
3	static void accessFiles() throws FileNotFoundException{	void, accessFiles, throws, FileNotFoundException	5	0	0	2	2	10	
4	try {								
5	FileReader f = new FileReader("D:\\Exceptions.doc");	FileReader, f , =, new, FileReader, "D:\\Exceptions.doc"	7	0	0	2	2	14	
6	System.out.println("File found");	System, ., out, ., println, "File found"	6	0	0	2	2	12	
	}								
7	catch(FileNotFoundException e) {	catch, FileNotFoundException, e	3	1	0	2	3	9	
8	System.out.println(e.getMessage());	System, ., out, ., println, e, ., getMessage	8	0	0	2	2	16	
9	throw e; // Rethrows the exception	throw, e	3	0	0	2	2	6	
	}								
	}								
10	public static void main(String[ ] args){	void, main, String, args	4	0	0	2	2	8	
11	try {								
12	accessFiles();	accessFiles	1	0	0	2	2	2	
	}								
13	catch(FileNotFoundException e) {	catch( ), FileNotFoundException, e	3	1	0	2	3	9	
14	System.out.println("File cannot be found!");	System, ., out, ., println, "File cannot be found!"	6	0	0	2	2	12	
	}								
	}								
	}								
		Cp							**98**