T.C.

SAKARYA UNIVERSITY

FACULTY OF COMPUTER AND INFORMATION SCIENCES

COMPUTER ENGINEERING DEPARTMENT

SOFTWARE TESTING ASSIGNMENT REPORT

OPERATOR OPERAND AND FUNCTION COUNTER PROGRAM

B181210562 - Hajer GAFSI

SAKARYA

April, 2023

Software Testing Course

Summary

The written program is a library that counts the number of operands, functions and operators splitting them into 5 categories numerical, logical, relational, double and single operators. All the written codes and classes should be tested using integration and unit tests.

Keywords: software testing, java, operators, JUnit5

1. DEVELOPED SOFTWARE

The code is made of 8 classes, the main class is called Document.

a. Analysis of Document Class

Table 1. Document class methods' screenshots and Analysis

Method	Explanation
`public Document(String documentPath) throws Exception`	Class Constructor, it takes the java file's path as a parameter, throws an exception if file is not a java file
`public String read() throws IOException {`	This method opens the file reads its content and assigns it as a string to the local variable "code"
`public void readAndCleanString() throws IOException`	This method calls the previous method then cleans the obtained string by removing the comments and ambiguous strings
`public void AnalyzeOperators() {` `EF = new ExpressionFinder(code);` `this.numberOfNumericOperators = EF.Analyze(EOperator.numerical);` `this.numberOfLogicalOperators = EF.Analyze(EOperator.logical);` `this.numberOfRelationalOperators = EF.Analyze(EOperator.relational);` `this.numberOfDoubleOperators = EF.Analyze(EOperator.doubleOp);` `this.numberOfSingleOperators = EF.Analyze(EOperator.single);` `}`	This method is responsible for counting the number of operators by category, it makes use of the ExpressionFinder class
`public void AnalyzeOperands() {` `OA = new OperandAnalyzer(code);` `this.numberOfOperands = OA.Analyze();` `}`	This method is responsible for counting the number of operands, it makes use of the operandAnalyzer class
`public void AnalyzeFunctions() {` `FD = new FunctionDetector(code);` `this.numberOfFunctions = FD.Analyze();` `}`	This method is responsible for counting the number of functions, it makes use of the FunctionDetector class

`public int getNumberOfNumericOperators() {`
`    return this.numberOfNumericOperators;`
`}`
 	
`public int getNumberOfLogicalOperators() {`
`    return this.numberOfLogicalOperators;`
`}`
 	
`public int getNumberOfRelationalOperators() {`
`    return this.numberOfRelationalOperators;`
`}`
`public int getNumberOfDoubleOperators() {`
`    return this.numberOfDoubleOperators;`
`}`
 	
`public int getNumberOfSingleOperators() {`
`    return this.numberOfSingleOperators;`
`}`
 	
`public int getNumberOfFunctions() {`
`    return this.numberOfFunctions;`
`}`
`public int getNumberOfOperands() {`
`    return this.numberOfOperands;`
`}`
	These methods are the classic get methods they give reading access to local private variables to the user

b. Other classes

Table 2. Other classes’ Analysis

Class	Explanation
`AmbiguousStringRemover`	This Class detects strings containing operators and replaces them with a letter-only string,this prevents the program from including operators inside strings into the total operator count.
`CommentFinder`	This Class detects comments containing operators and deletes them, this prevents the program from including operators inside comments into the total operator count.
`ExpressionFinder`	This class will find expressions containing one or more operators and return the count depending on the operator-type given as a parameter by making use of the OperatorFinder class, it also detects incrementing operators like – and ++.
`OperatorFinder`	This class counts the number of operators of a specific category present in the expression given to it as a parameter, it also detects incrementing operators.
`FunctionDetector`	This class detects functions present in a java code and returns total count it also eliminates special cases such as conditional statements and loops

`OperandAnalyzer`	This class counts the number of operands present in a java code it also takes into consideration single operand expressions like ++i.

c. Unit Tests

This Section treats some use cases of tests in the project

- Faker Library

`String fileName = faker.file().fileName();`	Using Faker to generate file names (with extensions than are likely to be other than .java to be later tested on Document() method
`faker.regexify("\s*(\\&\\&|\\\|\\\||!)\s*");`	Using Faker to generate a logical operator using the method regexify
`new Faker()).lorem().sentence(3) + new Faker().lorem().word()`	Using Faker to generate an operator-free string for testing on ExpressionFinder class

- Mockito Library

`Mockito.when(commentGenerator.generate()).thenReturn("/*********this is a comment**********/");`
`IGenerator commentGenerator = Mockito.mock(IGenerator.class);`
	Using Mockito to mock a comment-generator class' generate function

`IOperator operator = Mockito.mock(IOperator.class);`
`Mockito.when(operator.getOperator()).thenReturn((new Faker()).regexify(operatorRegex));`
`exp += operator.getOperator()`
	Using Mockito to mock the operator class' getOperator() function that generates an operator using faker and regex

`IGenerator oprandGenerator = Mockito.mock(IGenerator.class);`
`Mockito.when(oprandGenerator.generate()).thenReturn((new Faker()).lorem().sentence(3) + new Faker().lorem().word() + " " );`
	Using Mockito to mock a operand-generator class' generate function that returns an operator-free string