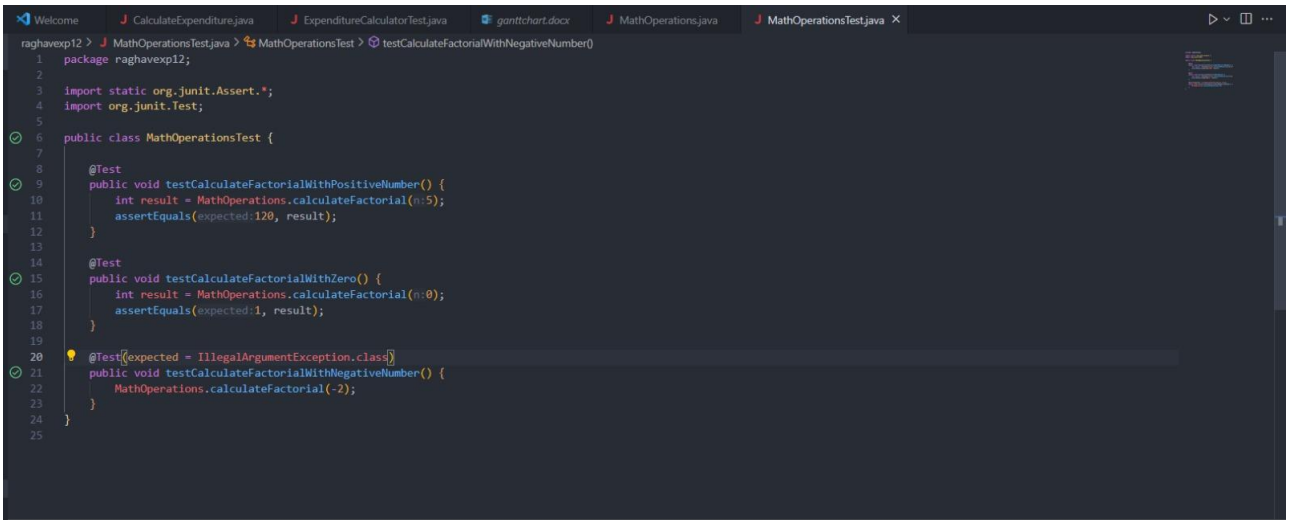
Java Code

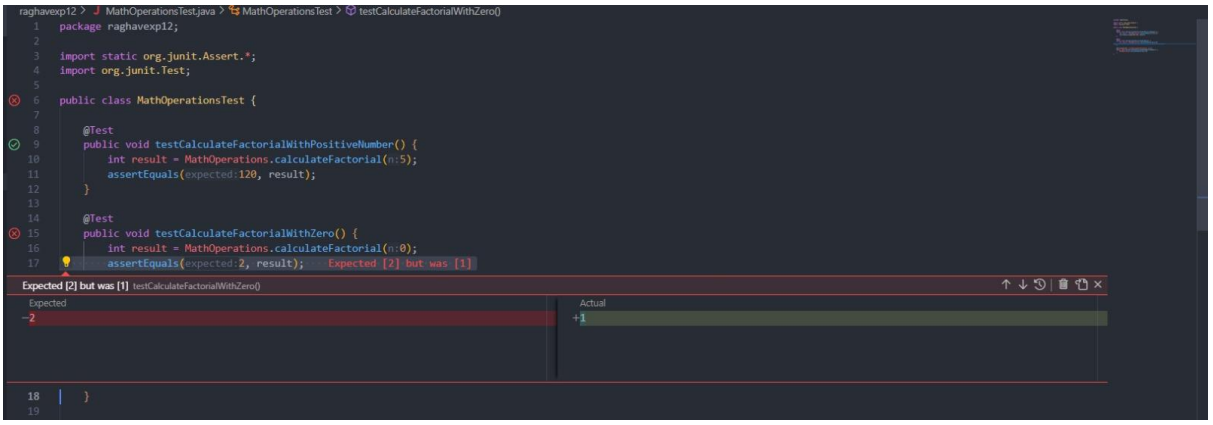
```
public class MathOperations {

    public static int calculateFactorial(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Input must be a non-negative integer.");
        } else {
            int result = 1;
            for (int i = 1; i <= n; i++) {
                result *= i;
            }
            return result;
        }
    }
}

```

```
import static org.junit.Assert.*;
import org.junit.Test;

public class MathOperationsTest {

    @Test
    public void testCalculateFactorialWithPositiveNumber() {
        int result = MathOperations.calculateFactorial(5);
        assertEquals(120, result);
    }

    @Test
    public void testCalculateFactorialWithZero() {
        int result = MathOperations.calculateFactorial(0);
        assertEquals(1, result);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testCalculateFactorialWithNegativeNumber() {
        MathOperations.calculateFactorial(-2);
    }
}
```

Testing



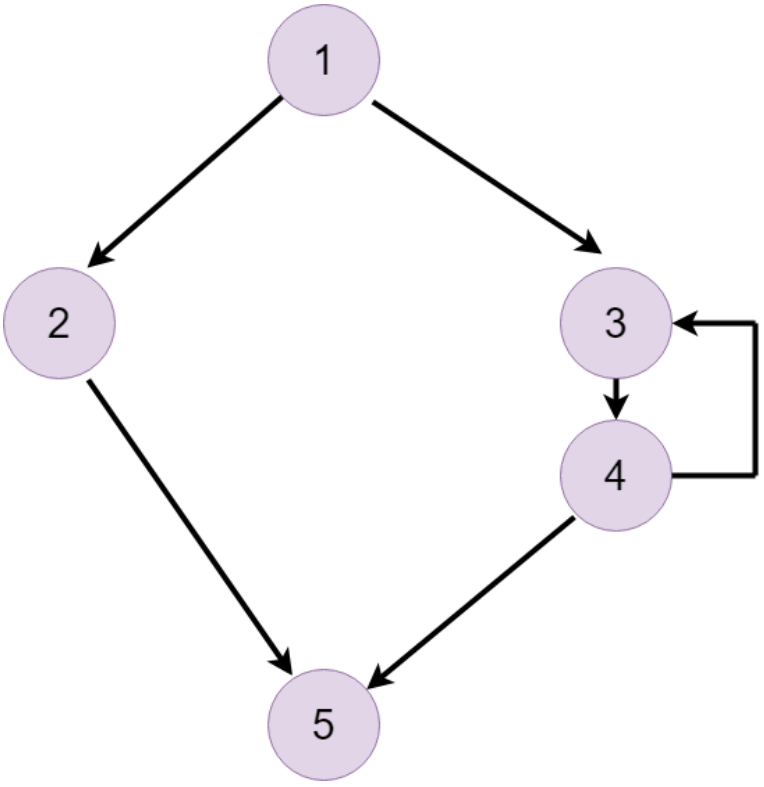
```
Welcome | CalculateExpenditure.java | ExpenditureCalculatorTest.java | ganttchart.docx | MathOperations.java | MathOperationsTest.java
raghavexp12 > MathOperationsTest.java > MathOperationsTest > testCalculateFactorialWithNegativeNumber()
1   package raghavexp12;
2
3   import static org.junit.Assert.*;
4   import org.junit.Test;
5
6   public class MathOperationsTest {
7
8       @Test
9       public void testCalculateFactorialWithPositiveNumber() {
10          int result = MathOperations.calculateFactorial(n:5);
11          assertEquals(expected:120, result);
12      }
13
14      @Test
15      public void testCalculateFactorialWithZero() {
16          int result = MathOperations.calculateFactorial(n:0);
17          assertEquals(expected:1, result);
18      }
19
20      @Test(expected = IllegalArgumentException.class)
21      public void testCalculateFactorialWithNegativeNumber() {
22          MathOperations.calculateFactorial(-2);
23      }
24  }
25
```



```
raghavexp12 > MathOperationsTest.java > MathOperationsTest > testCalculateFactorialWithZero()
1   package raghavexp12;
2
3   import static org.junit.Assert.*;
4   import org.junit.Test;
5
6   public class MathOperationsTest {
7
8       @Test
9       public void testCalculateFactorialWithPositiveNumber() {
10          int result = MathOperations.calculateFactorial(n:5);
11          assertEquals(expected:120, result);
12      }
13
14      @Test
15      public void testCalculateFactorialWithZero() {
16          int result = MathOperations.calculateFactorial(n:0);
17          assertEquals(expected:2, result);    Expected [2] but was [1]

Expected [2] but was [1] testCalculateFactorialWithZero()
Expected                    Actual
-2                          +1

18      }
19
```

Flow Graph

\



Cyclomatic Complexity for the above flowgraph

E(Number of Edges) = 5
N(Number of Nodes) = 4
P(No. of predicate Nodes) = 2
Cyclomatic complexity, V(G) = E-N+2
Therefore,
V(G) = 5-4+2 = 3

Or
V(G) = P + 1
V(G) = 2 + 1 = 3

Therefore, the cyclomatic complexity of the program code and it's drawn flowgraph is 3