CS731 Software Testing
Finance Calculators - Data Flow Coverage Testing

Instructor
Prof. Meenakshi D Souza

Team Members	Roll No.
Dishangkumar Patel	MT2022039
Jainam Shah	MT2022156

International Institute of Information And Technology, Bangalore
November 2023

---

1. Overview:

The goal of this project is to understand and perform practical aspects of testing. We have used Data Flow Coverage Criteria technique for testing the

source code that covers all def and all du path coverage and have used Junit as a testing tool.

Repo link : [letsFinance]

2. Project Statement:

letsFinance is a comprehensive Java terminal-based project designed to provide users with a set of powerful financial calculators to assist in various financial planning and investment decisions of their future. The suite includes following feature:

- Employee Provident Fund (EPF)
- Public Provident Fund (PPF)
- Systematic Investment Plan (SIP)
- Systematic Withdrawal Plan (SWP)
- Taxation
- Lumpsum
- Gratuity

3. Test Case Design Technique:

We have designed our test cases using **Data Flow Coverage Criteria** using **all defs** and **all du-path coverage.**

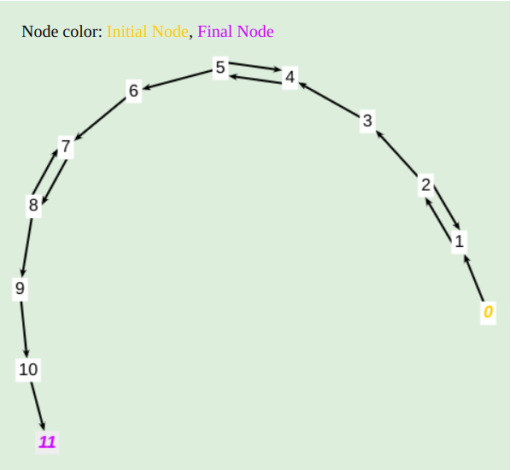
All Def Coverage:
For each def-path set S = du(n, v), TR contains at least one path d in S.

All DU-Path Coverage:
For each def-pair set S = du(ni , nj , v), TR contains every path d in S.

---

4. EMI Calculator Testing:

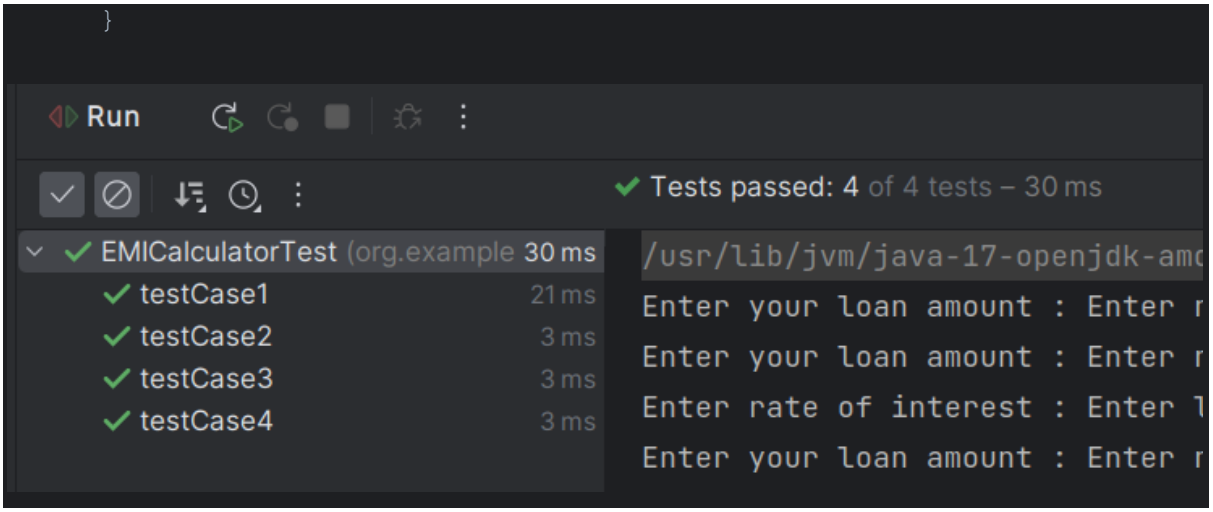
Data Flow Graph: The following is the data flow graph for the source code(refer the repo) and the corresponding def and du-path coverage.



EMI Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
val	{ 2, 5, 8 }	{ (2, 1), (2, 3), 3, (5, 4), (5, 6), 6, (8, 7), (8, 9), 9 }	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,9,10,11]
loanAmount	{ 3 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
interestRate	{ 6 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
loanTenure	{ 9 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	{ 10 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]

Test Cases:
Following are the test cases (passed) based on the above derived set of test paths.

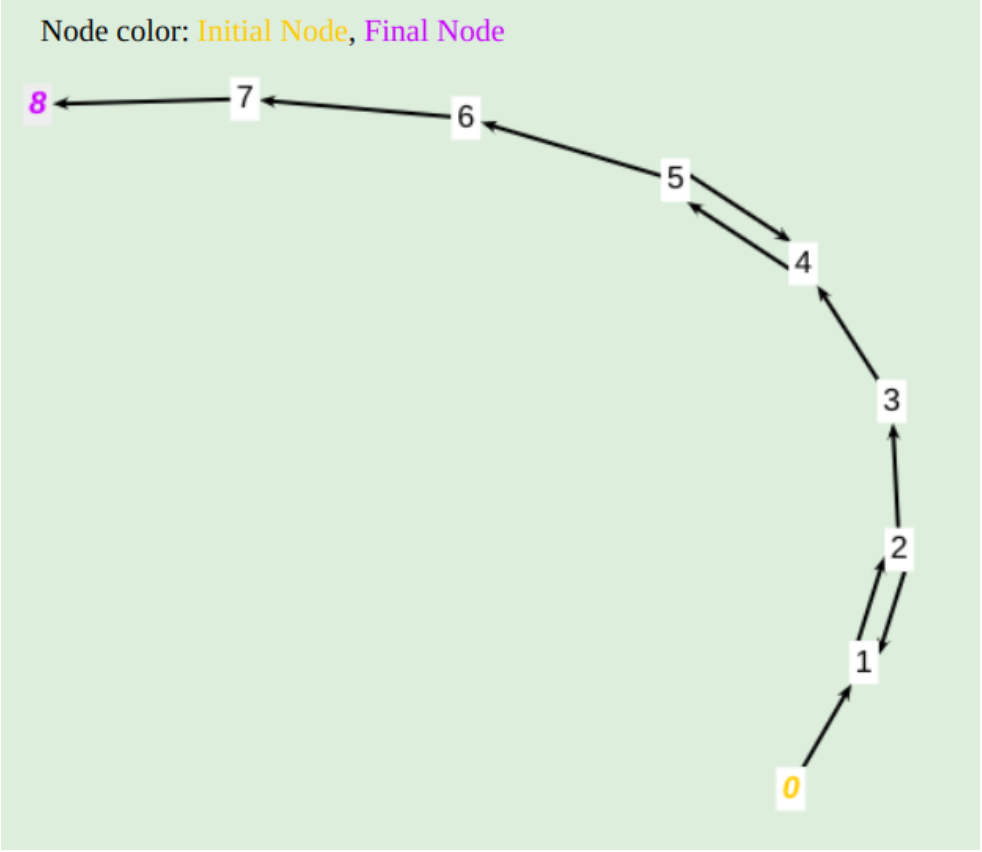
```
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class EMICalculatorTest {

    String input1 = "1000000\n5.5\n2\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11]
    String input2 = "2000000\n-5\n3.5\n2\n"; //
[0,1,2,3,4,5,4,5,6,7,8,9,10,11]
    String input3 = "2000000\n3.5\n-2\n2\n"; //
[0,1,2,3,4,5,6,7,8,7,8,9,10,11]
    String input4 = "-10000\n2000000\n3.5\n2\n"; //
[0,1,2,1,2,3,4,5,6,7,8,9,10,11]

    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        EMICalculator emiCalculator = new EMICalculator();
        Long actual = emiCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 44095L);
    }

    @Test
    public void testCase2(){
        testing(input2, 86405L);
    }
    @Test
    public void testCase3(){
        testing(input3, 86405L);
    }
    @Test
    public void testCase4(){
        testing(input4, 86405L);
    }
```



---

5. Gratuity Calculator Testing:

Data Flow Graph: The following is the data flow graph for the source code(refer the repo) and the corresponding def and du-path coverage.

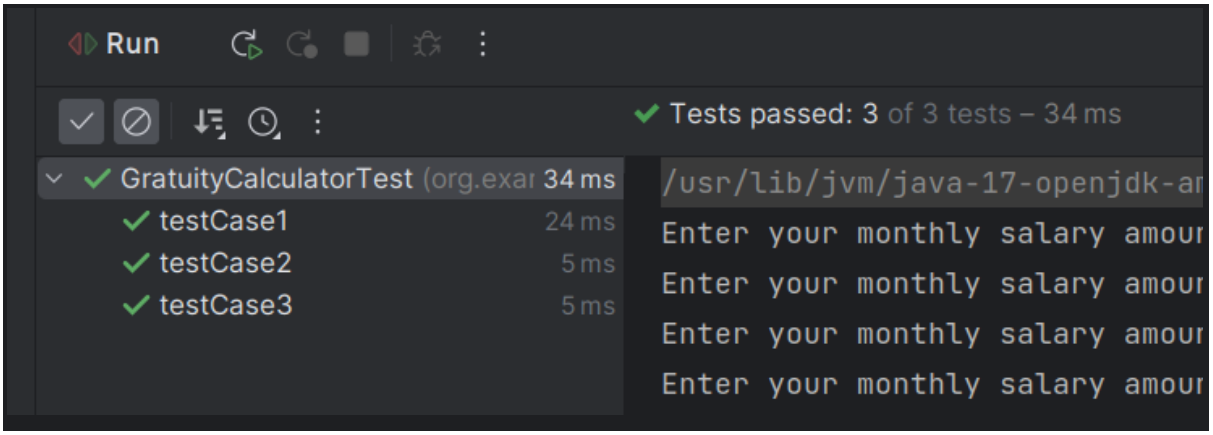


Gratuity Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
ms	{ 2 }	{ (2, 1), (2, 3), 3 }	[0,1,2,3,4,5,6,7,8]	[0,1,2,3,4,5,6,7,8], [0,1,2,1,2,3,4,5,6,7,8]
monthlySalary	{ 3 }	{ 7 }	[0,1,2,3,4,5,6,7,8]	[0,1,2,3,4,5,6,7,8]
yos	{ 5 }	{ (5, 4), (5, 6), 6 }	[0,1,2,3,4,5,6,7,8]	[0,1,2,3,4,5,6,7,8], [0,1,2,3,4,5,4,5,6,7,8]
yearOfServices	{ 6 }	{ 7 }	[0,1,2,3,4,5,6,7,8]	[0,1,2,3,4,5,6,7,8]
amount	{ 7 }	{ 8 }	[0,1,2,3,4,5,6,7,8]	[0,1,2,3,4,5,6,7,8]

Test Cases:
Following are the test cases (passed) based on the above derived set of test paths.

```java
package org.example;

import java.util.Scanner;

public class GratuityCalculator {
   private Double monthlySalary;
   private Double yearsOfService;

   public Double getMonthlySalary() {
       return monthlySalary;
   }

   public void setMonthlySalary(Double monthlySalary) {
       this.monthlySalary = monthlySalary;
   }

   public Double getYearsOfService() {
       return yearsOfService;
   }

   public void setYearsOfService(Double yearsOfService) {
       this.yearsOfService = yearsOfService;
   }

   public Long init(){
       try{
           Double ms, yos;
           Scanner scanner = new Scanner(System.in);

           while (true) {
```

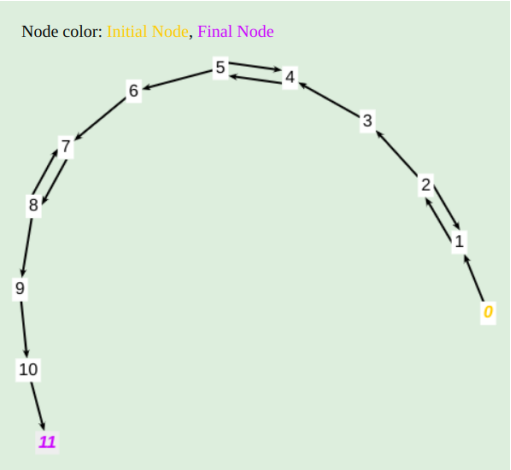
```
                System.out.print("Enter your monthly salary amount
: ");
                ms = scanner.nextDouble();
                if (ms >= 0) {
                    break;
                }
                System.out.println("Please enter positive monthly
salary : ");
            }
            setMonthlySalary(ms);

            while (true) {
                System.out.print("Enter years of service : ");
                yos = scanner.nextDouble();
                if (yos >= 0) {
                    break;
                }
                System.out.println("Please enter valid year of
service");
            }
            setYearsOfService(yos);

            Long totalValue = calculateReturn();
            System.out.println("You are eligible for " + totalValue
+ " gratuity");
            return totalValue;
        } catch (Exception e){
            return -1L;
        }
    }

    public Long calculateReturn(){
        Double amnt = getYearsOfService() * getMonthlySalary() * 15
/ 26;
        return Math.min(1000000, amnt.longValue());
    }

}
```



---

6. Lumpsum Calculator Testing:

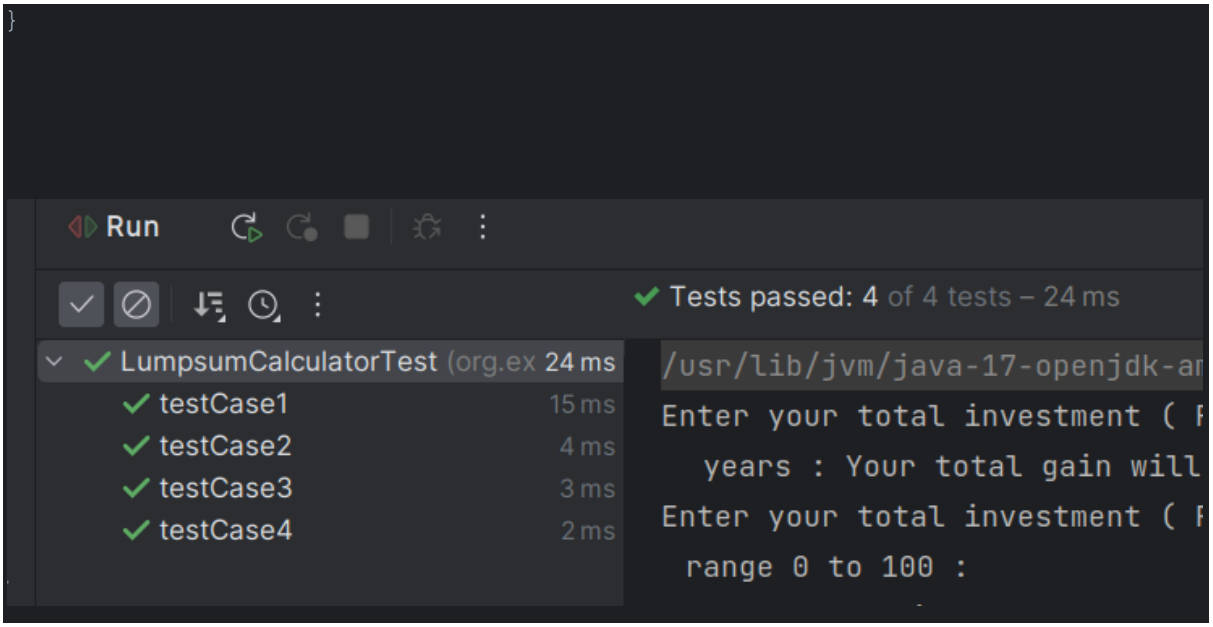
Data Flow Graph: The following is the data flow graph for the source code(refer the repo) and the corresponding def and du-path coverage.



Lumpsum Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
val	{ 2, 5, 8 }	{ (2, 1), (2, 3), 3, (5, 4), (5, 6), 6, (8, 7), (8, 9), 9 }	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,9,10,11]
principleAmount	{ 3 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
interestRate	{ 6 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
timePeriod	{ 9 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	{ 10 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]

Test Cases:
Following are the test cases (passed) based on the above derived set of test paths.

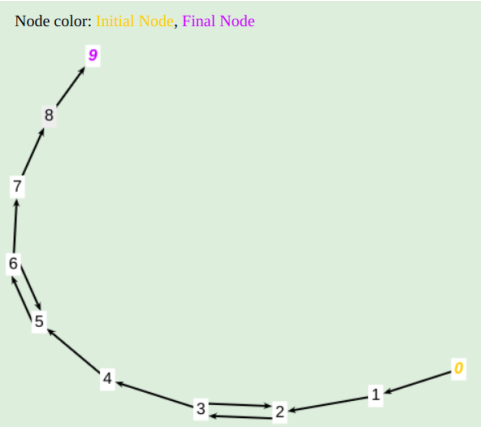
```
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class LumpsumCalculatorTest {
   String input1 = "1000000\n5.5\n2\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11]
   String input2 = "2000000\n-5\n3.5\n2\n"; //
[0,1,2,3,4,5,4,5,6,7,8,9,10,11]
   String input3 = "2000000\n3.5\n-2\n2\n"; //
[0,1,2,3,4,5,6,7,8,7,8,9,10,11]
   String input4 = "-10000\n2000000\n3.5\n2\n"; //
[0,1,2,1,2,3,4,5,6,7,8,9,10,11]

   public void testing(String input, Long expectedTax){
       ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(input.getBytes());
       System.setIn(byteArrayInputStream);
       LumpsumCalculator lumpsumCalculator = new
LumpsumCalculator();
       Long actual = lumpsumCalculator.init();
       Assert.assertEquals(expectedTax,actual);
   }

   @Test
   public void testCase1(){
       testing(input1, 1113025L);
   }

   @Test
   public void testCase2(){
       testing(input2, 2142449L);
   }
   @Test
   public void testCase3(){
       testing(input3, 2142449L);
   }

   @Test
   public void testCase4(){
       testing(input4, 2142449L);
   }
```



---

7. PPF Calculator Testing:

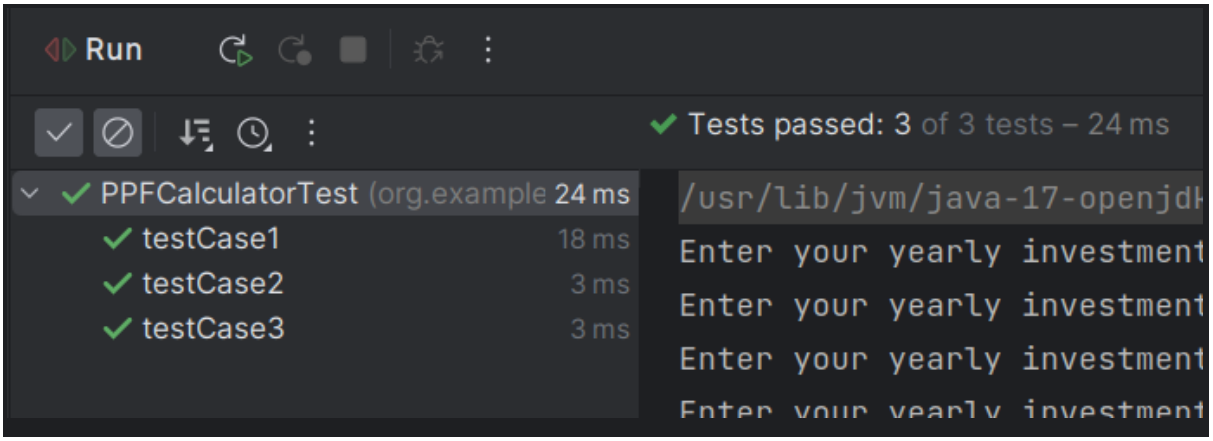
Data Flow Graph: The following is the data flow graph for the source code(refer the repo) and the corresponding def and du-path coverage.



PPF Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
yi	{ 3 }	{ (3, 2), (3, 4), 4 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9], [0,1,2,3,2,3,4,5,6,7,8,9]
tp	{ 6 }	{ (6, 5), (6, 7), 7}	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9], [0,1,2,3,4,5,6,5,6,7,8,9]
yearlyInvestment	{ 4 }	{ 8 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9]
timePeriod	{ 7 }	{ 8 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9]
rateOfInterest	{ 1 }	{ 8 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9]
amount	{ 8 }	{ 9 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9]

Test Cases:
Following are the test cases (passed) based on the above derived set of test paths.

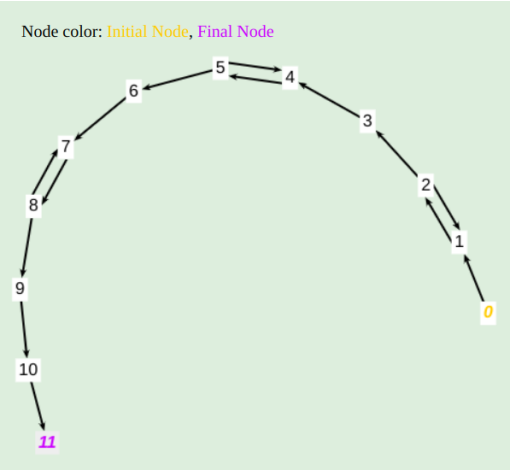
```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class PPFCalculatorTest {

   String input1 = "100000\n2\n"; // [0,1,2,3,4,5,6,7,8,9]
   String input2 = "-10000\n200000\n2\n"; //
[0,1,2,3,2,3,4,5,6,7,8,9]
   String input3 = "200000\n-2\n2\n"; // [0,1,2,3,4,5,6,5,6,7,8,9]

   public void testing(String input, Long expectedTax){
       ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(input.getBytes());
       System.setIn(byteArrayInputStream);
       PPFCalculator ppfCalculator = new PPFCalculator();
       Long actual = ppfCalculator.init();
       Assert.assertEquals(expectedTax,actual);
   }

   @Test
   public void testCase1(){
       testing(input1, 207099L);
   }

   @Test
   public void testCase2(){
       testing(input2, 414199L);
   }

   @Test
   public void testCase3(){
       testing(input3, 414199L);
   }

}
```



---

8. SIP Calculator Testing:

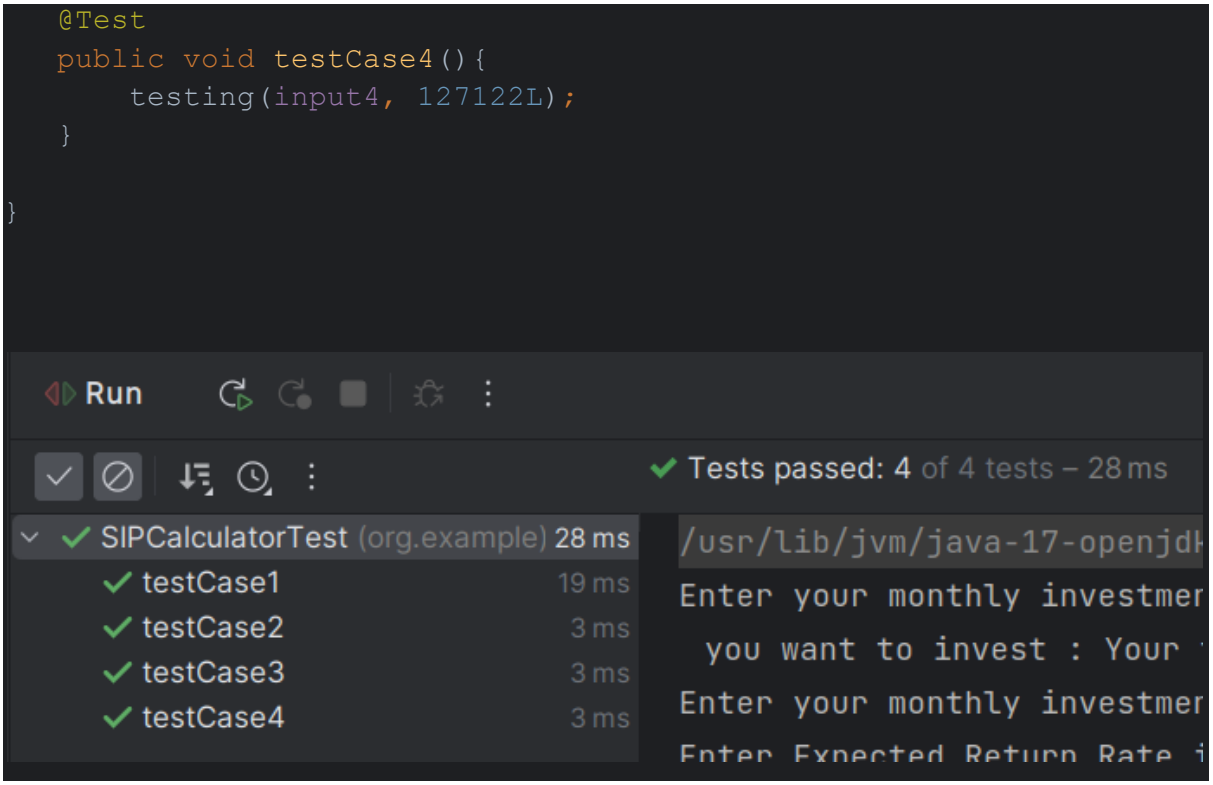
Data Flow Graph: The following is the data flow graph for the source code(refer the repo) and the corresponding def and du-path coverage.



SIP Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
val	{ 2, 5, 8 }	{ (2, 1), (2, 3), 3, (5, 4), (5, 6), 6, (8, 7), (8, 9), 9 }	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,9,10,11]
monthlyInvestment	{ 3 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
expectedReturnRateInPercentage	{ 6 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
timePeriodInYear	{ 9 }	{ 10 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	{ 10 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11]	[0,1,2,3,4,5,6,7,8,9,10,11]

Test Cases:
Following are the test cases (passed) based on the above derived set of test paths.

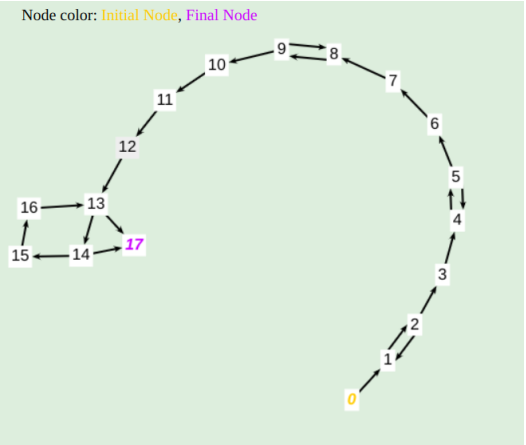
```
package org.example;

import org.junit.Assert;
import org.junit.Test;

import java.io.ByteArrayInputStream;

public class SIPCalculatorTest {

   String input1 = "3500\n5.5\n2\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11]
   String input2 = "5000\n-5\n5.5\n2\n"; //
[0,1,2,3,4,5,4,5,6,7,8,9,10,11]
   String input3 = "5000\n5.5\n-2\n2\n"; //
[0,1,2,3,4,5,6,7,8,7,8,9,10,11]
   String input4 = "-10000\n5000\n5.5\n2\n"; //
[0,1,2,1,2,3,4,5,6,7,8,9,10,11]


   public void testing(String input, Long expectedTax){
       ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(input.getBytes());
       System.setIn(byteArrayInputStream);
       SIPCalculator sipCalculator = new SIPCalculator();
       Long actual = sipCalculator.init();
       Assert.assertEquals(expectedTax,actual);
   }

   @Test
   public void testCase1(){
       testing(input1, 88985L);
   }

   @Test
   public void testCase2(){
       testing(input2, 127122L);
   }

   @Test
   public void testCase3(){
       testing(input3, 127122L);
   }
```



---

9. SWP Calculator Testing:

Data Flow Graph: The following is the data flow graph for the source code(refer the repo) and the corresponding def and du-path coverage.



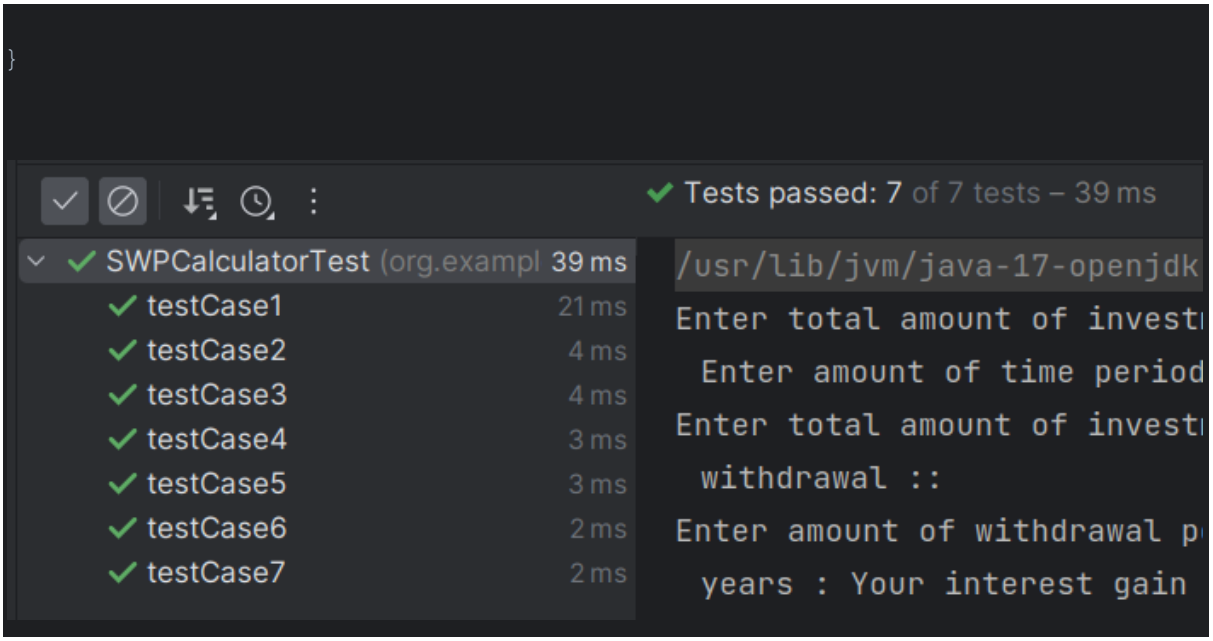
SWP Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
val	{ 2, 5, 7, 9 }	{ (2, 1), (2, 3), 3, (5, 4), (5, 6), 6, 7, (9, 8), (9, 10), 10 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,8,9,10,11,12,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,8,9,10,11,12,13,17]
totalInvestment	{ 3 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
withdrawalAmount	{ 6 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
expectedReturnRate	{ 7 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
timePeriod	{ 10 }	{ 11 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
deduct	{ 11 }	{ 14 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17]

val1	{ 11, 14, 15 }	{ 14, (14, 17), (14, 15), 15 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]
gain	{ 11, 15 }	{ 15 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]
n	{ 11 }	{ (13, 14), (13, 17) }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
i	{ 12, 16 }	{ (13, 14), (13, 17), 16 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]
returnAmnt	{ 17 }	{ 17 }	No Path needed	No Path needed
tmp	{ 15 }	{ 15 }	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]

Test Cases:
Following are the test cases (passed) based on the above derived set of test paths.

```
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class SWPCalculatorTest {
   String input1 = "100000\n5000\n5.5\n0\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
   String input2 = "100000\n-5000\n5000\n5.5\n3\n"; //
[0,1,2,3,4,5,4,5,6,7,8,9,10,11,12,13,17]
   String input3 = "100000\n5000\n5.5\n-2\n3\n"; //
[0,1,2,3,4,5,6,7,8,9,8,9,10,11,12,13,17]
   String input4 = "-3500\n100000\n5000\n5.5\n3\n"; //
[0,1,2,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
   String input5 = "500000\n500000\n5.5\n2\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17]
   String input6 = "500000\n250000\n5.5\n2\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]
```

```
    String input7 = "600000\n200000\n5.5\n2\n"; //
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15,16,13,17]



    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        SWPCalculator swpCalculator = new SWPCalculator();
        Long actual = swpCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 0L);
    }

    @Test
    public void testCase2(){
        testing(input2, 4621L);
    }

    @Test
    public void testCase3(){
        testing(input3, 4621L);
    }

    @Test
    public void testCase4(){
        testing(input3, 4621L);
    }

    @Test
    public void testCase5(){
        testing(input3, 4621L);
    }

    @Test
    public void testCase6(){
        testing(input3, 4621L);
    }

    @Test
    public void testCase7(){
        testing(input3, 4621L);
    }
```


}
Tests passed: 7 of 7 tests – 39 ms
SWPCalculatorTest (org.exampl 39 ms
testCase1 21 ms
testCase2 4 ms
testCase3 4 ms
testCase4 3 ms
testCase5 3 ms
testCase6 2 ms
testCase7 2 ms
/usr/lib/jvm/java-17-openjdk
Enter total amount of investı
Enter amount of time period
Enter total amount of investı
withdrawal ::
Enter amount of withdrawal p
years : Your interest gain