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**

- **Individual Contribution Matrix:**

Sr. no	Modules	Contributor	Type of Contribution
1	Employee Provident Fund (EPF)	Jainam	Business Logic, Code, TestCase design, TestCase Impl
2	Public Provident Fund (PPF)	Jainam	Business Logic, Code, TestCase design, TestCase Impl
3	Systematic Investment Plan (SIP)	Jainam	Business Logic, Code, TestCase design, TestCase Impl
4	Systematic Withdrawal Plan (SWP)	Jainam	Business Logic, Code, TestCase design, TestCase Impl
5	Taxation	Dishang	Business Logic, Code, TestCase design, TestCase Impl
6	Lumpsum	Dishang	Business Logic, Code, TestCase design, TestCase Impl
7	Gratuity	Dishang	Business Logic, Code, TestCase design, TestCase Impl

- **Table Of Content:**

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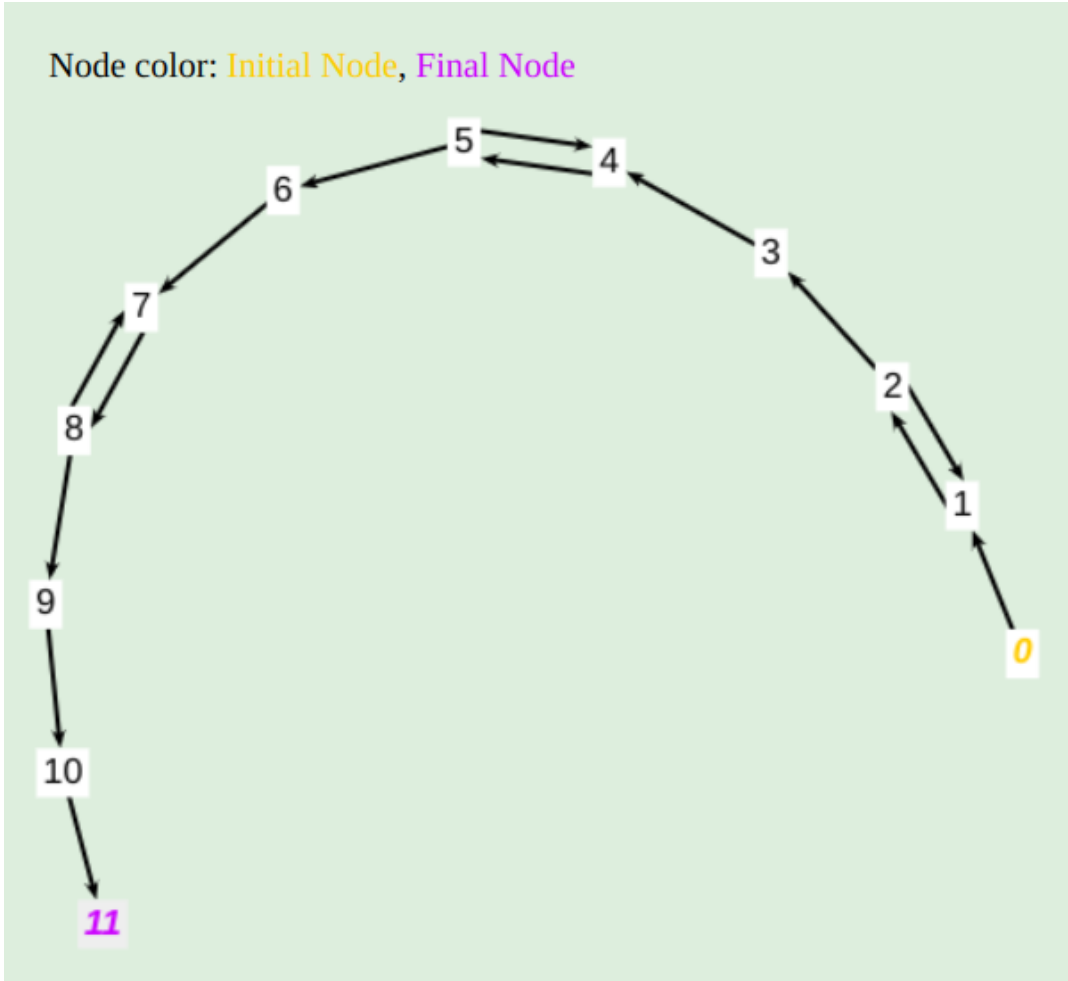
All Def Coverage:
    For each def-path set S = du(n, v), TR contains at least one path d in S.

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.

```java
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());
```

```java
            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);
        }

    }
```

▶ **Run**   ⟳ ⟲ ■ ⟳ ⋮

✓ ⊘ ⤓ ⧖ ⋮   ✓ Tests passed: 4 of 4 tests – 30 ms

⌄ ✓ EMICalculatorTest (org.example 30 ms   /usr/lib/jvm/java-17-openjdk-amd
   ✓ testCase1   21 ms   Enter your loan amount : Enter r
   ✓ testCase2   3 ms
   ✓ testCase3   3 ms   Enter your loan amount : Enter r
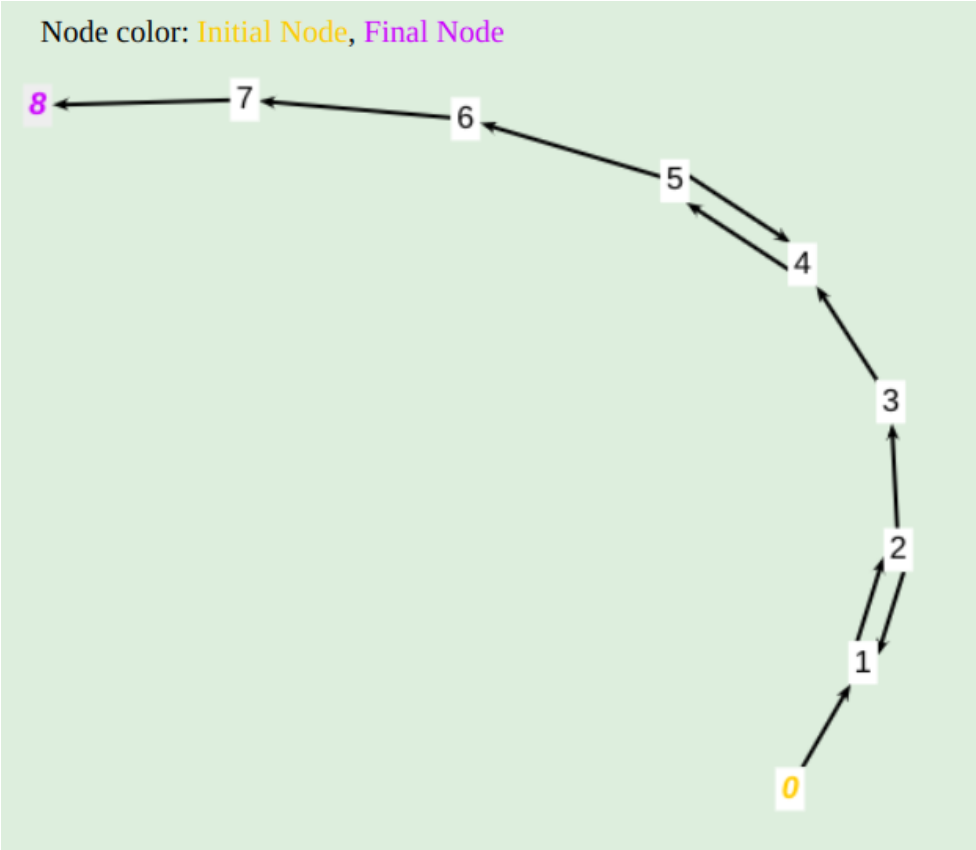   ✓ testCase4   3 ms   Enter rate of interest : Enter l

                                     Enter your loan amount : Enter r

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.

Node color: Initial Node, Final Node

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;
```

```java
            }
            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());
}
}
}
```

◀▷ **Run**   G↻ G↻ ■ ⚙ ⋮

✓ ⊘ | ⬇≡ ⏱ ⋮   ✔ Tests passed: 3 of 3 tests – 34 ms

∨ ✔ GratuityCalculatorTest (org.exan 34 ms   /usr/lib/jvm/java-17-openjdk-ar
    ✔ testCase1   24 ms   Enter your monthly salary amoun
    ✔ testCase2   5 ms   Enter your monthly salary amoun
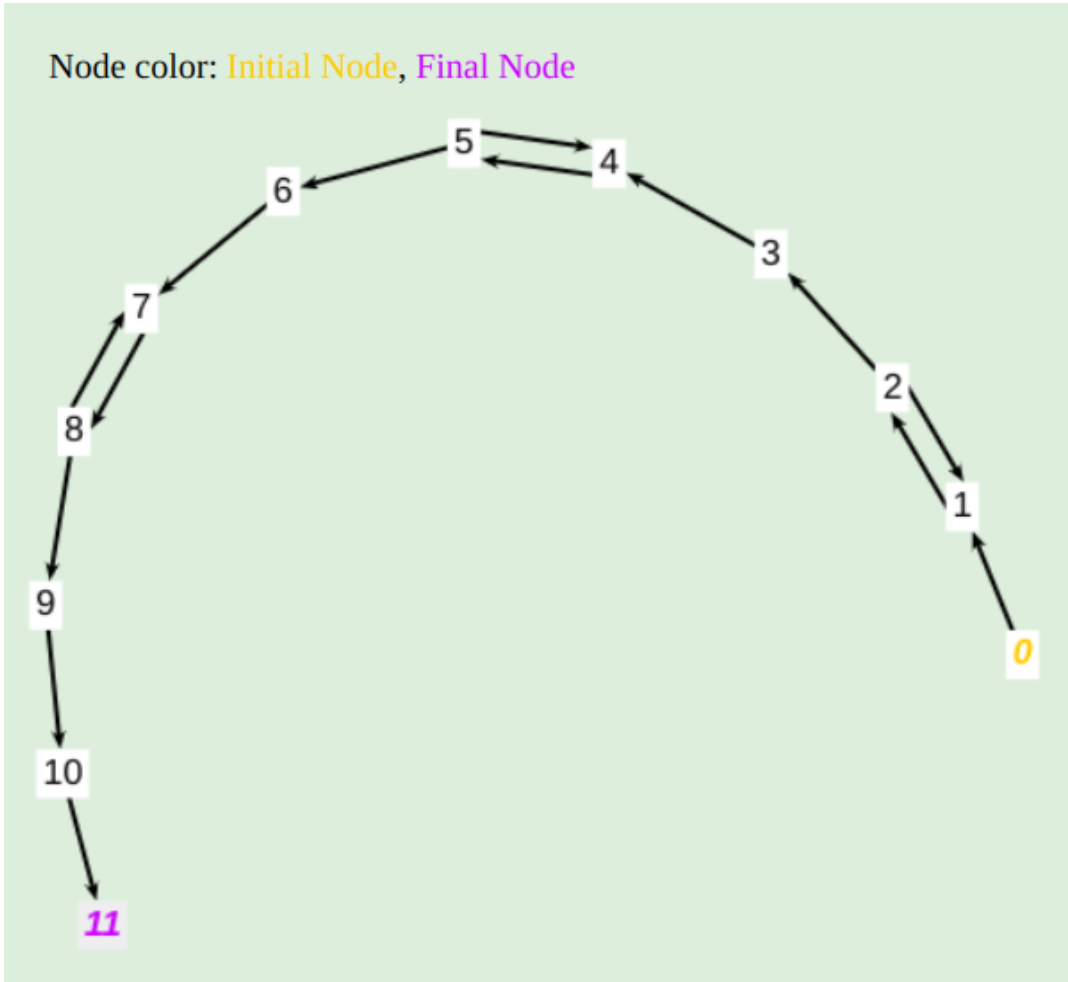   ✔ testCase3   5 ms   Enter your monthly salary amoun
                                                Enter your monthly salary amoun

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.

```java
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(){
```

```java
        testing(input1, 1113025L);
    }

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

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

◀▷ **Run**   ⟳ ⟳ ■ 🐞 ⋮

✓ ⊘ ⇩ ⏱ ⋮   ✓ Tests passed: **4** of 4 tests – 24 ms

∨ ✓ LumpsumCalculatorTest (org.ex **24 ms**   /usr/lib/jvm/java-17-openjdk-ar
    ✓ testCase1   15 ms   Enter your total investment (F
    ✓ testCase2   4 ms   years : Your total gain will
    ✓ testCase3   3 ms   Enter your total investment (F
    ✓ testCase4   2 ms   range 0 to 100 :

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.

Node color: Initial Node, Final Node
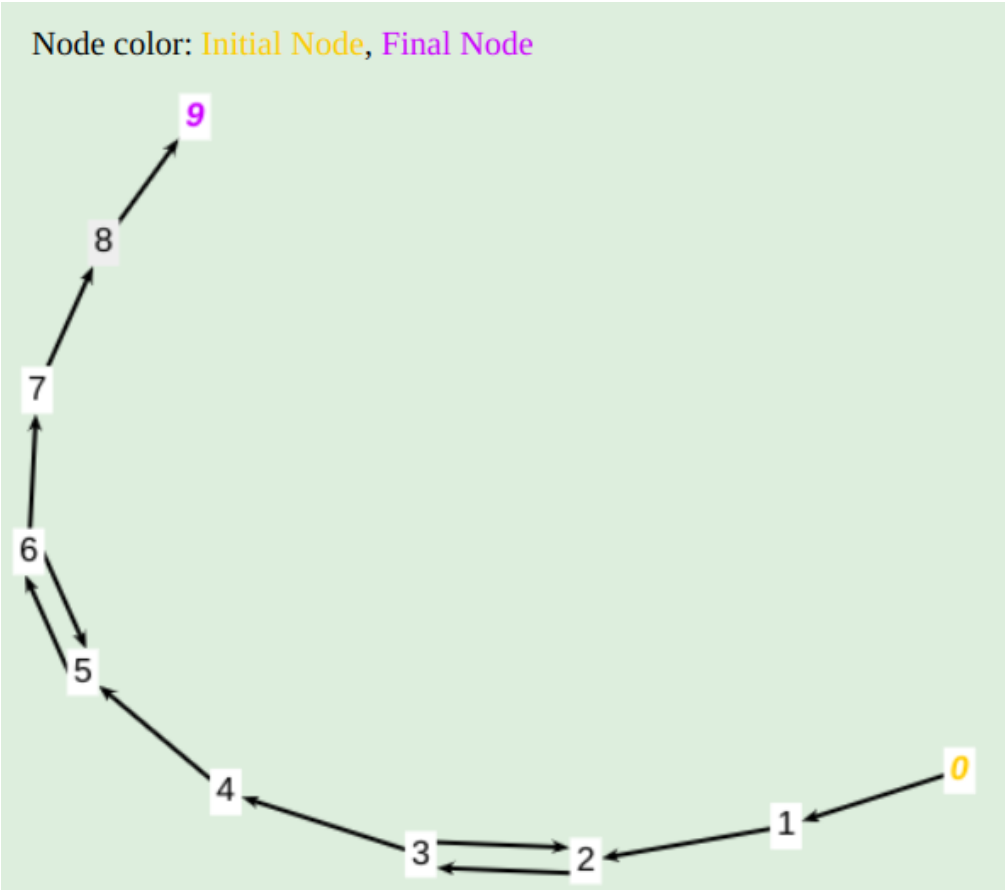
PPF Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
yi	{ 3 }	{ (3, 2), (3, 4), 4 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9], [0,1,2,3,2,3,4,5,6,7,8,9]
tp	{ 6 }	{ (6, 5), (6, 7), 7 }	[0,1,2,3,4,5,6,7,8,9]	[0,1,2,3,4,5,6,7,8,9], [0,1,2,3,4,5,6,5,6,7,8,9]
yearlyInvestment	{ 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);
    }

}
```

Run

✓ Tests passed: **3** of 3 tests – 24 ms

✓ PPFCalculatorTest (org.example 24 ms

/usr/lib/jvm/java-17-openjdk

✓ testCase1   18 ms

Enter your yearly investment

✓ testCase2   3 ms

Enter your yearly investment

✓ testCase3   3 ms
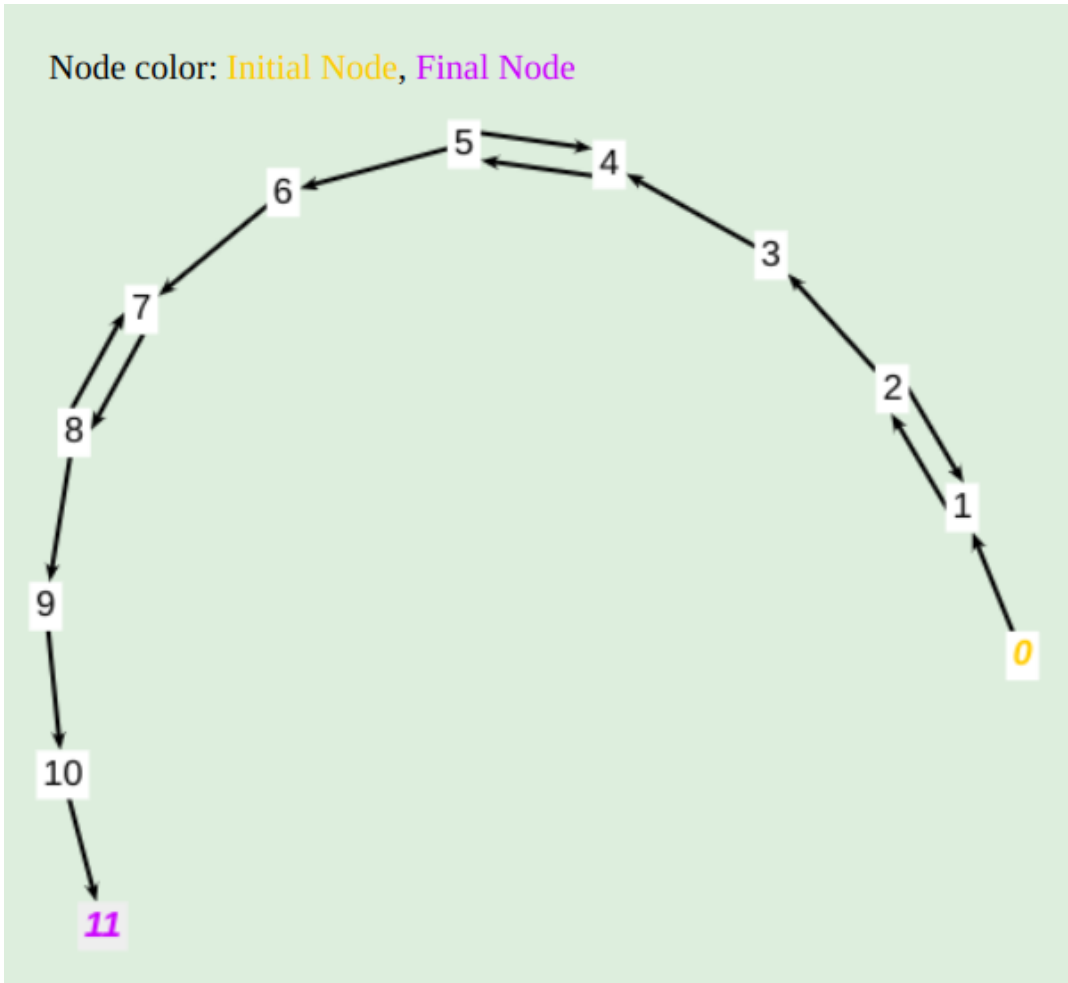
Enter your yearly investment

Enter your yearly investment

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.

Node color: Initial Node, Final Node

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.

```java
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();
```

```java
            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);
    }

    @Test
    public void testCase4(){
        testing(input4, 127122L);
    }

}
```

Run

✓ Tests passed: 4 of 4 tests – 28 ms

```
✓ SIPCalculatorTest (org.example) 28 ms      /usr/lib/jvm/java-17-openjdk
    ✓ testCase1                    19 ms   Enter your monthly investmen
    ✓ testCase2                     3 ms    you want to invest : Your
    ✓ testCase3                     3 ms   Enter your monthly investmen
    ✓ testCase4                     3 ms   Enter Expected Return Rate i
```

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.

Node color: Initial Node, Final Node

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.

```java
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(){
```

```java
        testing(input4, 4621L);
    }

    @Test
    public void testCase5(){
        testing(input5, 4621L);
    }

    @Test
    public void testCase6(){
        testing(input6, 4621L);
    }

    @Test
    public void testCase7(){
        testing(input7, 4621L);
    }

}
```

✓ Tests passed: 7 of 7 tests – 39 ms

✓ SWPCalculatorTest (org.exampl **39 ms**	/usr/lib/jvm/java-17-openjdk
✓ testCase1 21 ms	Enter total amount of investr
✓ testCase2 4 ms	Enter amount of time period
✓ testCase3 4 ms	Enter total amount of investr
✓ testCase4 3 ms	withdrawal ::
✓ testCase5 3 ms	Enter amount of withdrawal p
✓ testCase6 2 ms	years : Your interest gain
✓ testCase7 2 ms	

10. Income Tax Calculator:-

Data Flow Graph: The following is the data flow graph for the source code of income tax calculator (refer the repo) and the corresponding def and du-path coverage.
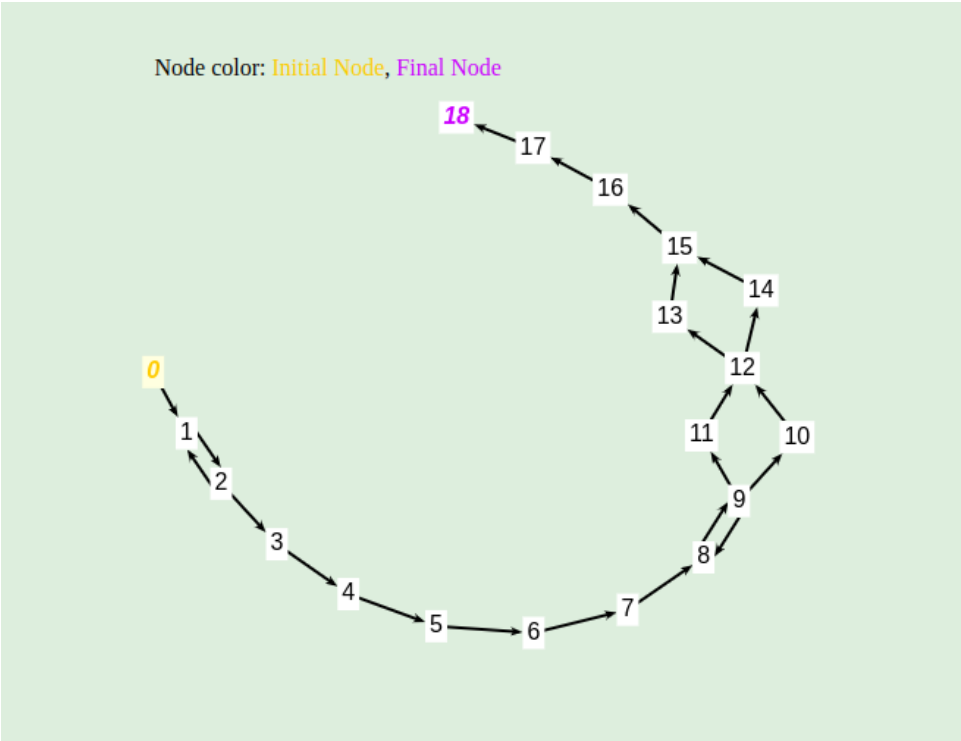
Foreign Company's Incometax
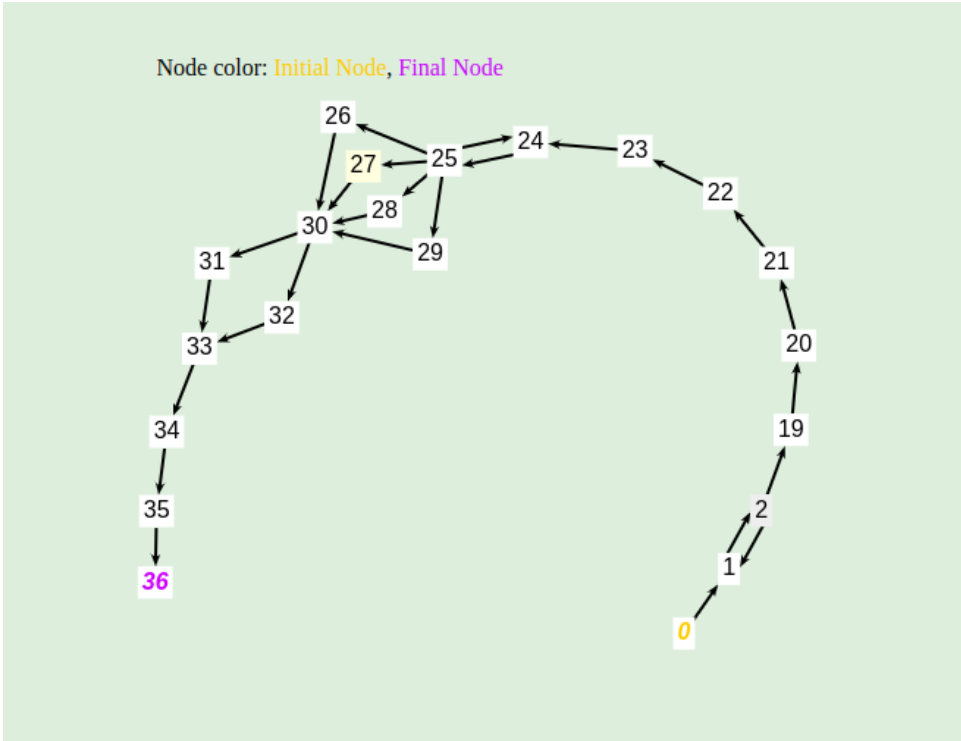
Domestic Companies' Incometax
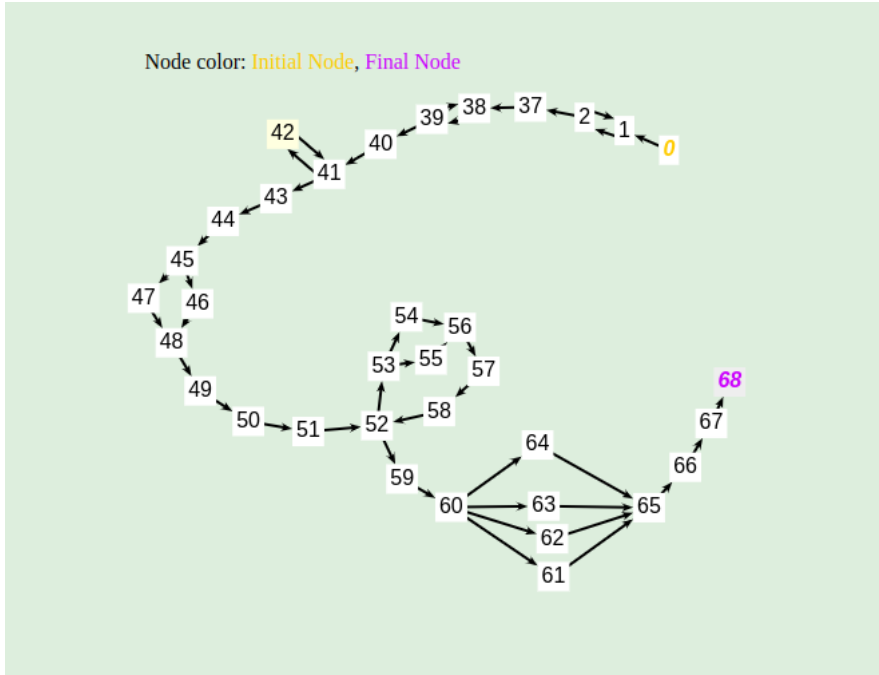
Salaried Employee's Incometax

Income Tax Calculator				
Variables	Definitions	Uses	All Def Coverage	All DU Path Coverage
slabs	{ 51 }	{ (52,52), (53,64), (53,55), 56 }	{ [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68], [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,59,60,61,65,66,67,68], }	{ [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68], [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,59,60,61,65,66,67,68] }
slab	{ 1 }	{ 51 }	{ [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68] }	{ [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68], [0,1,2,37,38,39,40,41,42,43,44,45,47,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68] }
type	{ 2 }	{ (2,1), (2,3), (2,19), (2,37) }	{ [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68] }	{ [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68] , [0,1,2], [0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18], [0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36] }

income	{ 3, 19, 44 }	{ 4, 7, 20, 23, 50 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68],
				[0,1,2,37,38,39,40,41,42,43,44,45,47,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
				}

ngoDonation	{ 4, 20 }	{ 5. 21 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				}

goDonation	{ 4, 20 }	{ 5. 21 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				}

netQualifyingLimit	{ 4, 20 }	{ 5. 21 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				}

deduction	{ 5, 21, 48 }	{ 6, 22, 49 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
				}

deductedAmt	{ 6, 22, 49 }	{ 7, 23, 50 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
				}

taxableAmt	{ 7, 23, 50, 57 }	{ 10, 11, (12,13), (12,14), 26, 27, 28, 29, (30,31), (30,32), (52,53), 54, 55, 57, (60,61), (60,62), (60,63), (60,64) }	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18]
			}
				{[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
				[0,1,2,1,2,3,4,5,6,7,8,9,11,12,14,15,16,17,18],
				[0,1,2,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18],
				
				[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,26,30,32,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,27,30,32,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,27,30,31,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,28,30,32,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,28,30,31,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,29,30,32,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,29,30,31,33,34,35,36],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68],
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,62,65,66,67,68],
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54

				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,64,65,66,67,68], [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,63,65,66,67,68], [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,62,65,66,67,68], [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,61,65,66,67,68], }

id	{ 9, 25 }	{ (9,8), (9,10), (9,11), (25,24), (25,26), (25,27), (25,28), (25,29) }	{
			[0,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18],
			[0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36],
			}
				{
				[0,1,2,3,4,5,6,7,8,9,8,9,10,12,13,15,16,17,18],
				[0,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
				
				[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,27,30,31,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,28,30,31,33,34,35,36],
				[0,1,2,19,20,21,22,23,24,25,29,30,31,33,34,35,36]
				
				}

| ta | { 10, 11, 26, 27, 28, 29, 56 } | { 12, 30, 56, 59 } | {

[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18]

[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,27,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,28,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,29,30,31,33,34,35,36]

[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
} | {
[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
[0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18]

[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,27,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,28,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,29,30,31,33,34,35,36]

[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68],
} |

| taxAmt | { 12, 30, 59 } | { 13, 14, 16, 18, 31, 32, 34, 36, 61, 62, 63, 64, 66, 68 } | {

[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],

[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],

[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68]

} | {
[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
[0,1,2,1,2,3,4,5,6,7,8,9,11,12,14,15,16,17,18],

[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],

[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,62,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,63,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],

} |

| Sc | { 13, 14, 61, 62, 63, 64, 31, 32 } | { 15, 33, 65 } | {

[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],

[0,1,2,1,2,3,4,5,6,7,8,9,11,12,14,15,16,17,18],

[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],

[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,62,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,63,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]

} | {

[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],

[0,1,2,1,2,3,4,5,6,7,8,9,11,12,14,15,16,17,18],

[0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36],
[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],

[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,62,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,63,65,66,67,68],
[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]

} |

Surcharge	{ 15,33,65 }	{ 18, 36, 68 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
			
			[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
			
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
				
				[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

charge	{ 16, 34, 60 }	{ 17, 35, 67 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
			
			[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
			
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
				
				[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

healthAndEduCess	{ 17, 35, 67 }	{ 18, 36, 68 }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
			
			[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
			
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
				
				[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

NetTax	{ 18, 36, 68 }	{ x,y,z }	{
			[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
			
			[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
			
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18],
				
				[0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

schemeId	{ 39 }	{ (39,38), 40 }	{
			[0,1,2,37,38,39,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,37,38,39,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

regimeId	{ 40 }	{ (41,42), (45,46), (41,43), (45,47) }	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				[0,1,2,37,38,39,40,41,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

ageGrp	{ 40, 42 }	{ (42,41), 43 }	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
			[0,1,2,37,38,39,40,41,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,37,38,39,40,41,42,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				[0,1,2,37,38,39,40,41,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

ageGrpId	{ 43 }	{ 51 }	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				[0,1,2,37,38,39,40,41,42,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				}

80c	{ 45, 46 }	{ 48 }	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
				}

80ccd1b	{ 45, 46, 47 }	{ 48 }	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
			[0,1,2,37,38,39,40,41,42,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
			}
				{
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				[0,1,2,37,38,39,40,41,42,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				}

i	{51, 58}	{(52,53), (52,59), (53,54), (53,55), 55, 56, 58}	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68],
			}
				{
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,64,65,66,67,68],
				
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68],
				
				}

minA	{54, 55}	{56, 57}	{
			[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68]
			}
				{
				[0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68]
				}

```java
package org.example;

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

public class TaxCalculatorTest{

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
    String input1 = "1\n1\n1\n8000000\n10000\n25000\n1000\n2000\n6788\n";

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,59,60,61,65,66,67,68]
    String input2 = "1\n1\n1\n8000000\n10000\n25000\n1000\n2000\n6788\n";

    //  [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    String input3 = "1\n1\n1\n80000000\n10000\n25000\n1000\n2000\n6788\n";

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
    String input4 = "1\n1\n1\n8000000\n10000\n25000\n1000\n2000\n6788\n";

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,62,65,66,67,68]
    String input5 = "1\n1\n1\n15000000\n10000\n25000\n1000\n2000\n6788\n";

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,63,65,66,67,68]
    String input6 = "1\n1\n1\n40000000\n10000\n25000\n1000\n2000\n6788\n";

    // [0,1,2,37,38,39,40,41,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    String input7 = "1\n2\n80000000\n10000";

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68]
    String input8 = "1\n1\n1\n70000000\n10000\n25000\n1000\n2000\n6788\n";

    // [0,1,2,37,38,39,40,41,42,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    String input9 = "1\n1\n-1\n1\n70000000\n10000\n25000\n1000\n2000\n6788\n";



    // Following test-paths are not possible practically based on business logic, although they are possible theoretically
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,64,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,64,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,63,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,62,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,61,65,66,67,68]
```

```java
// [0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18]
String input10 = "-1\n3\n80000000\n200000\n300000\n1\n";

// [0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18]
String input11 = "-1\n3\n80000000\n200000\n300000\n0\n";

// [0,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18]
String input12 = "3\n80000000\n200000\n300000\n1\n";

// [0,1,2,3,4,5,6,7,8,9,8,9,10,12,13,15,16,17,18]
String input13 = "3\n80000000\n200000\n300000\n-1\n1\n";

// [0,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18]
String input14 = "3\n80000000\n200000\n300000\n0\n";

// [0,1,2,1,2,3,4,5,6,7,8,9,11,12,14,15,16,17,18]
String input15 = "-1\n3\n150000000\n200000\n300000\n1\n";
// [0,1,2,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18]

// [0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36]
String input16 = "2\n70000000\n200000\n300000\n1\n";

// [0,1,2,19,20,21,22,23,24,25,27,30,31,33,34,35,36]
String input17 = "2\n70000000\n200000\n300000\n2\n";

// [0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36]
String input18 = "2\n70000000\n200000\n300000\n-1\n1\n";

// [0,1,2,19,20,21,22,23,24,25,28,30,31,33,34,35,36]
String input19 = "2\n70000000\n200000\n300000\n3\n";

// [0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36]
String input20 = "2\n150000000\n200000\n300000\n-1\n1\n";

// [0,1,2,19,20,21,22,23,24,25,29,30,31,33,34,35,36]
String input21 = "2\n70000000\n200000\n300000\n4\n";

// [0,1,2,19,20,21,22,23,24,25,26,30,32,33,34,35,36]
String input22 = "2\n250000000\n200000\n300000\n1\n";

// [0,1,2,19,20,21,22,23,24,25,27,30,32,33,34,35,36]
String input23 = "2\n250000000\n200000\n300000\n2\n";
```

```java
// [0,1,2,19,20,21,22,23,24,25,28,30,32,33,34,35,36]
String input24 = "2\n250000000\n200000\n300000\n3\n";

// [0,1,2,19,20,21,22,23,24,25,29,30,32,33,34,35,36]
String input25 = "2\n250000000\n200000\n300000\n4\n";


public void testing(String input, int expectedTax){
    ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
    System.setIn(byteArrayInputStream);
    TaxCalculator taxCalculator = new TaxCalculator();
    int netTax = taxCalculator.init().intValue();
    Assert.assertEquals(expectedTax,netTax);
}

@Test
public void testCase1(){
    testing(input1, 2506932);
}
@Test
public void testCase2(){
    testing(input2, 2506932);
}
@Test
public void testCase3(){
    testing(input3, 33556679);
}
@Test
public void testCase4(){
    testing(input4, 2506932);
}
@Test
public void testCase5(){
    testing(input5, 5115885);
}
@Test
public void testCase6(){
    testing(input6, 15220792);
}
@Test
public void testCase7(){
    testing(input7, 33465645);
}
```

```java
    @Test
    public void testCase8(){
        testing(input8, 29326679);
    }
    @Test
    public void testCase9(){
        testing(input9, 29326679);
    }
    @Test
    public void testCase10(){
        testing(input10, 42135000);
    }
    @Test
    public void testCase11(){
        testing(input11, 33708000);
    }
    @Test
    public void testCase12(){
        testing(input12, 42135000);
    }
    @Test
    public void testCase13(){
        testing(input13, 42135000);
    }
    @Test
    public void testCase14(){
        testing(input14, 33708000);
    }
    @Test
    public void testCase15(){
        testing(input15, 81477500);
    }
    @Test
    public void testCase16(){
        testing(input16, 19286250);
    }
    @Test
    public void testCase17(){
        testing(input17, 16971900);
    }
    @Test
    public void testCase18(){
        testing(input18, 19286250);
    }
    @Test
```

```java
    public void testCase19(){
        testing(input19, 11571750);
    }
    @Test
    public void testCase20(){
        testing(input20, 43355000);
    }
    @Test
    public void testCase21(){
        testing(input21, 23143500);
    }
    @Test
    public void testCase22(){
        testing(input22, 72355000);
    }
    @Test
    public void testCase23(){
        testing(input23, 63672400);
    }
    @Test
    public void testCase24(){
        testing(input24, 43413000);
    }
    @Test
    public void testCase25(){
        testing(input25, 86826000);
    }
}
}
```

Run

✓ Tests passed: 25 of 25 tests – 83 ms

```
✓ TaxCalculatorTest (org.examp 83 ms    /usr/lib/jvm/java-17-openjdk-amd64/bin/java ...
  ✓ testCase1            21 ms           Select type:
  ✓ testCase2             6 ms           1.Individual
  ✓ testCase3             3 ms           2.Domestic Company
  ✓ testCase4             3 ms           3.Foreign Company
  ✓ testCase5             3 ms           0.To Exit
  ✓ testCase6             3 ms           Select you regime:
  ✓ testCase7             2 ms           1.Old Tax Regime
  ✓ testCase8             2 ms           2.New tax Regime
  ✓ testCase9             2 ms
  ✓ testCase10            3 ms
  ✓ testCase11            4 ms
```