Software Testing Fall 2019

NOVEMBER 14

Submitted By:

Rajeev Pankaj Shukla (MT2018091)

Ravindra Singh Pawar (MT2018093)

Contents

About the Application

The application features unit converter with factorial and combinations as additional features.

The unit converter comes up with conversion in four categories - weight, area, length and temperature. Each of these categories in included with subcategories given as follows:

1) Weight: milligram, centigram, gram, kilogram, decigram, ton, pound, ounce.
2) Area: sq. Millimeter, sq. Centimeter, sq. Meter, sq. Meter, sq. Kilometer, sq. Acre, sq. Hectare.
3) Length: nanometer, millimeter, centimeter, meter, kilometer, inch, foot, yard, mile.
4) Temperature: Celsius, kelvin, Fahrenheit.

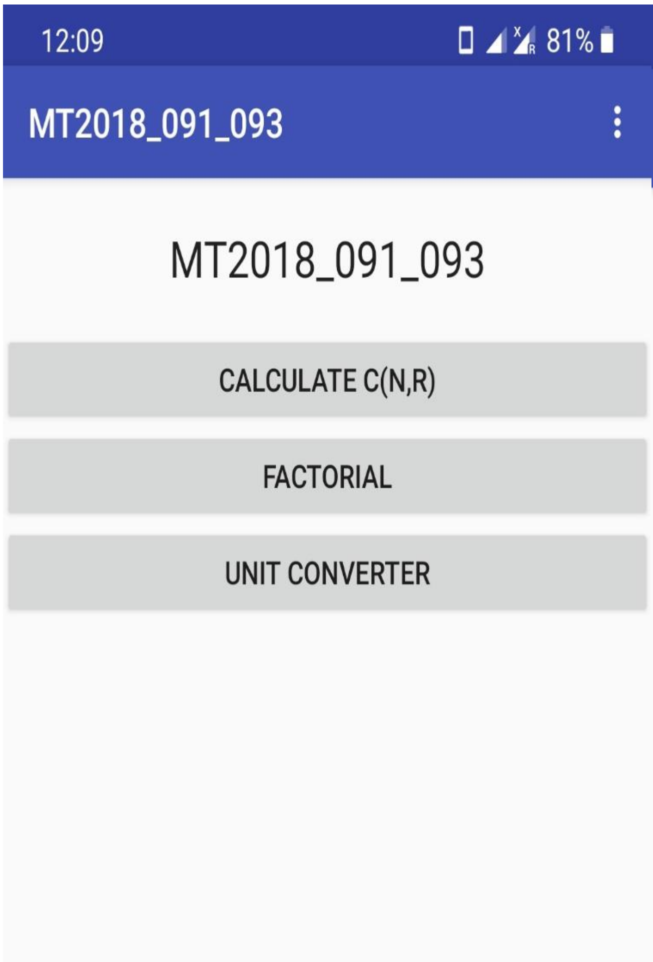
The factorial features calculation of factorial with a given numeric value. The interface is provided with inbuilt digit buttons to enter numeric value.

The combination features calculation of mathematical value of C(n, r) after entering numeric value of n and r.

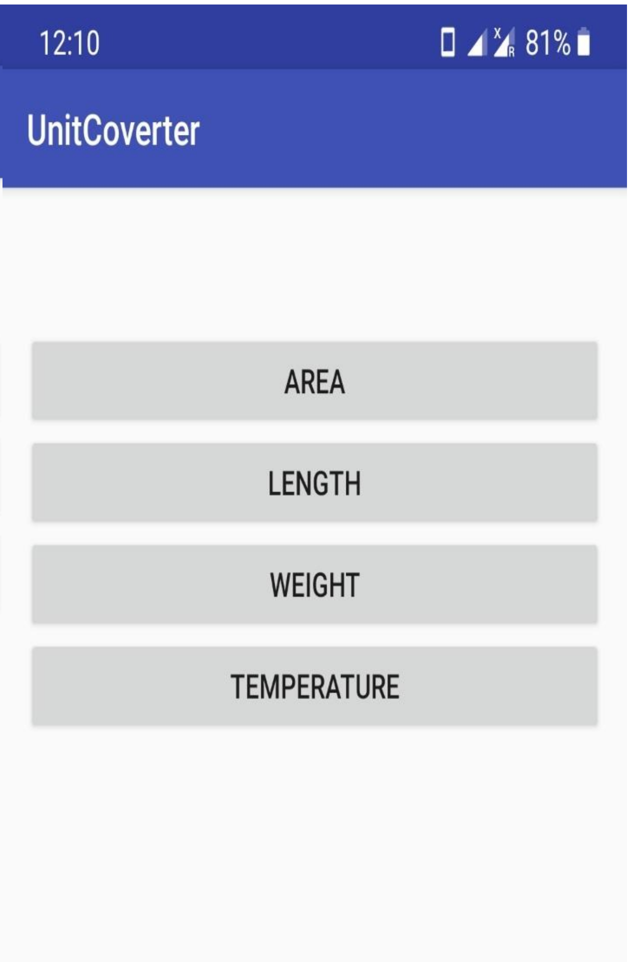
Git Repository:
https://github.com/RajeevPankajShukla/SoftwareTesting.git
Features:

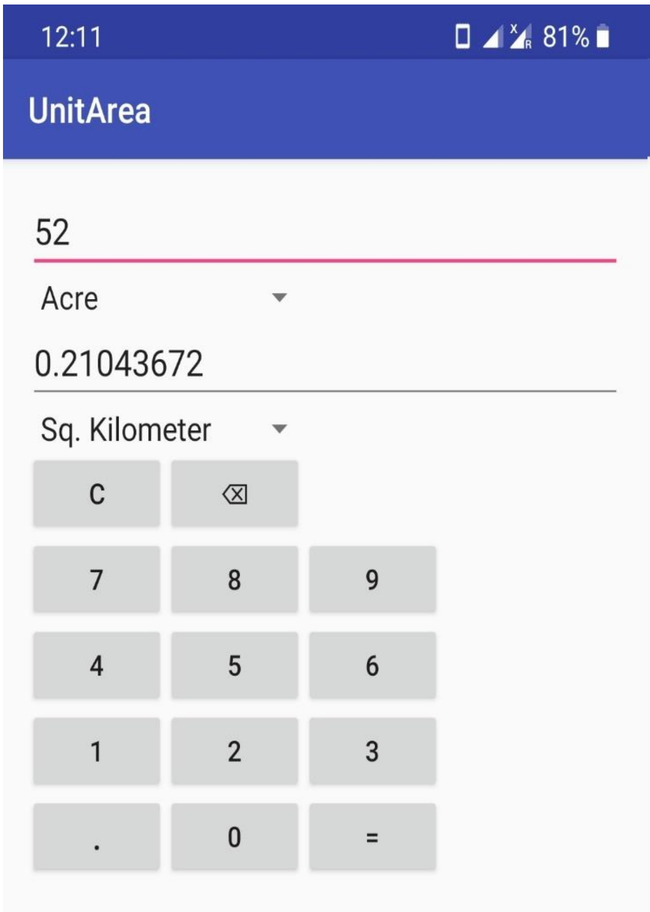
- Easy to use and light weight application.
- Beautiful, simple and stylish design.
- Backspace button to delete the last digit to correct a simple mistake.

Screenshots of our application are added on next page.



MT2018_091_093



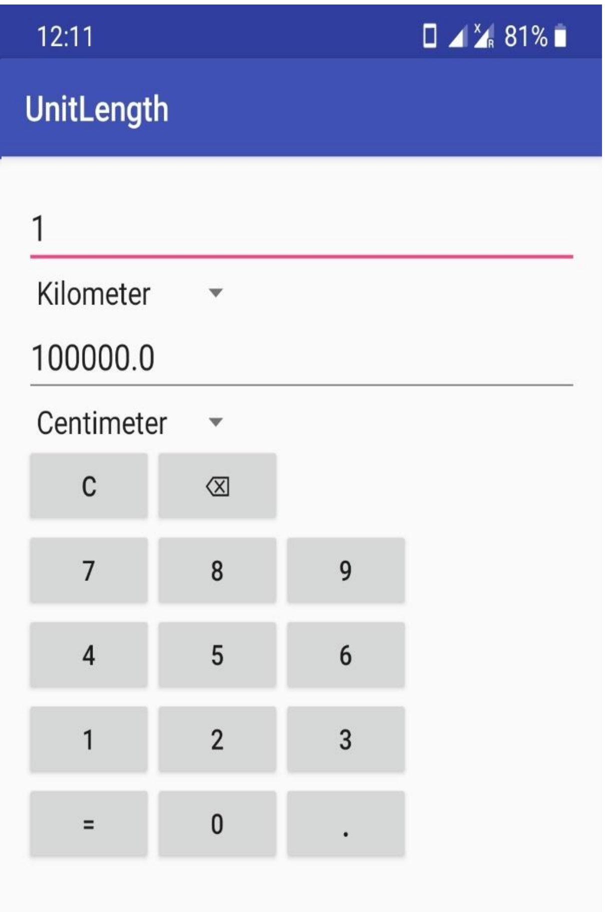


52

Acre

0.21043672

Sq. Kilometer

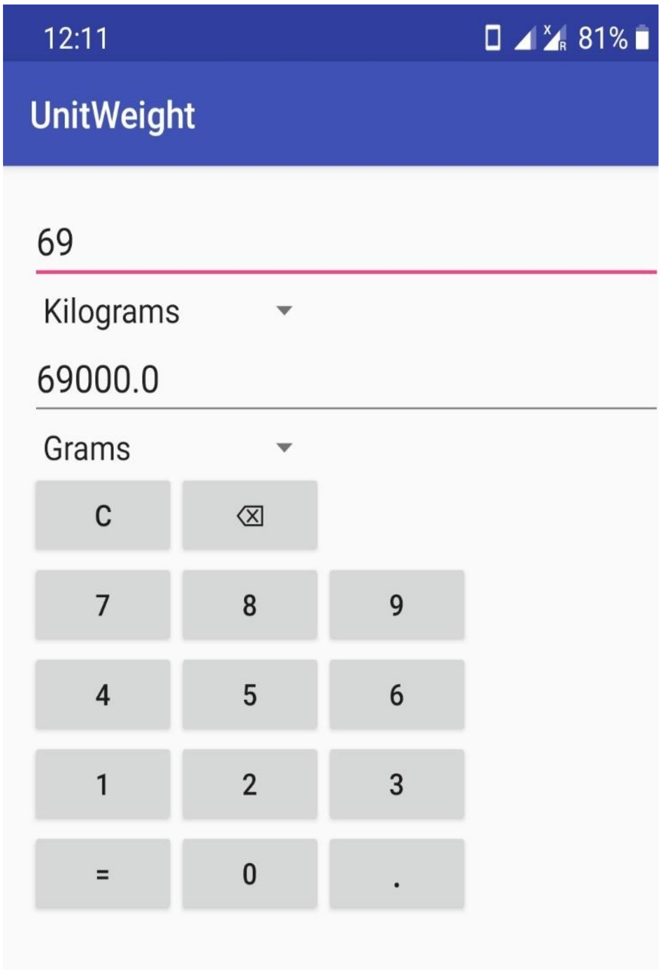


1

Kilometer

100000.0

Centimeter



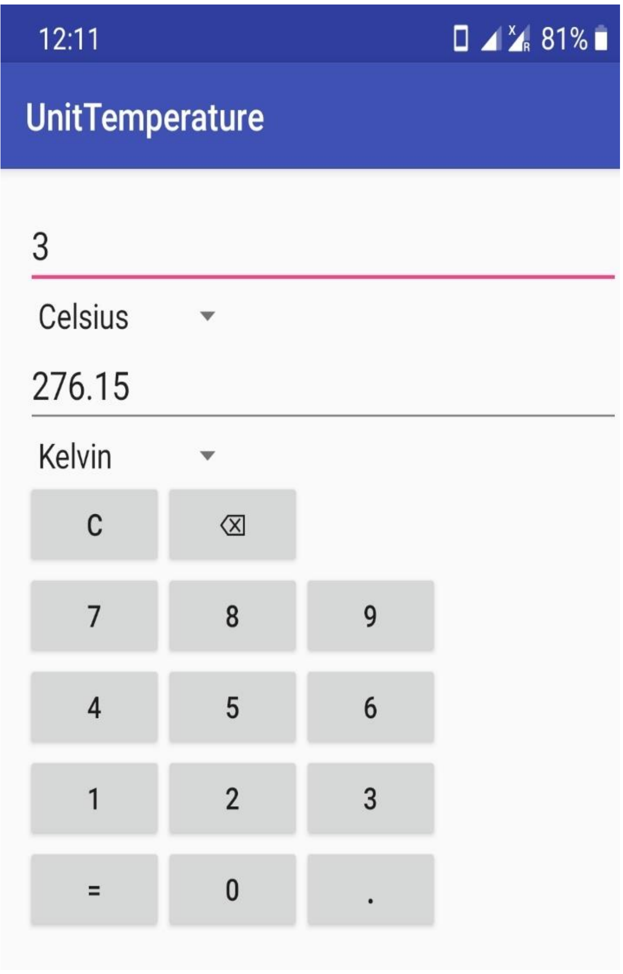
UnitWeight

69

Kilograms

69000.0

Grams



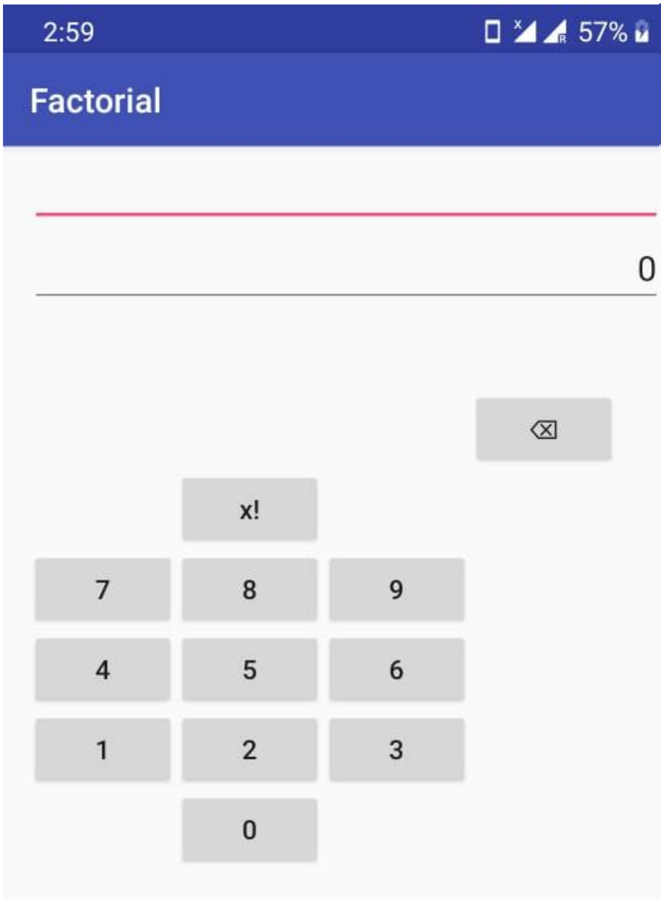
UnitTemperature

3

Celsius

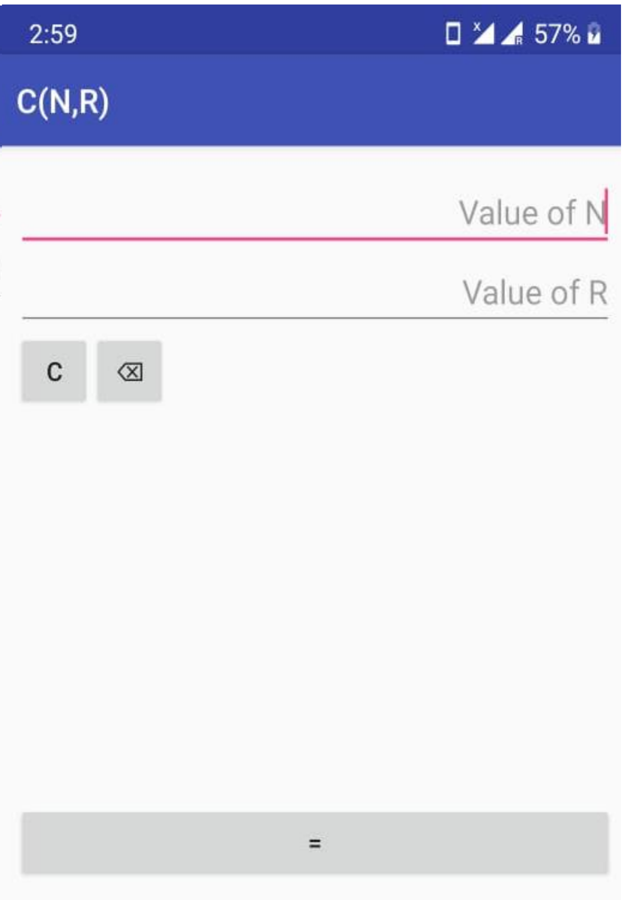
276.15

Kelvin



Factorial

0



C(N,R)

Value of N

Value of R

Problem Statement

To design test cases for Edge Coverage and for Prime Paths Coverage represented by using Control Flow Graph (CFG).

Control Flow Graph

A Control Flow Graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside of a program unit.

Elements of control flow graph:

1) Nodes: Statements or sequences of statements (basic blocks).
2) Edges: Transfers of control.

Basic Block: A sequence of statements such that if the first statement is executed, all statements will be (no branches).

Edge Coverage

Test requirement TR, contains each reachable path of length up to 1, inclusive, in graph G. By path of length it means that it allows edge coverage for graphs with one node and no edges.

Prime Path Coverage

A prime path is a simple path that does not appear as a proper sub-path of any other simple path. The test requirement TR, contains each prime path in graph G.

Testing the Application

Control Flow Graph

Tools Used: Understand scitools (https://scitools.com/)

Control flow graphs of functions under testing are added below.

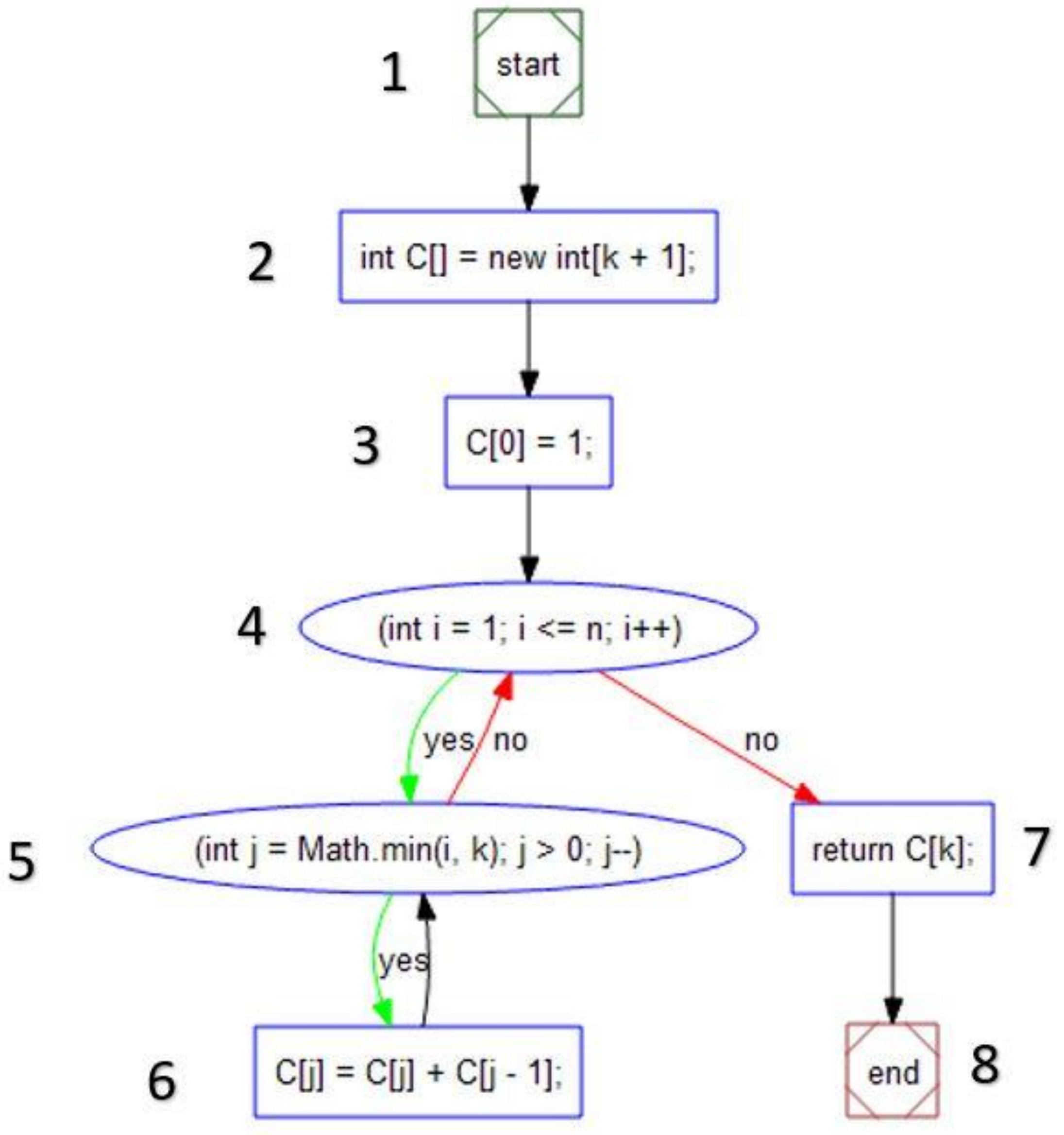


Figure 1: Combination(n,r) CFG

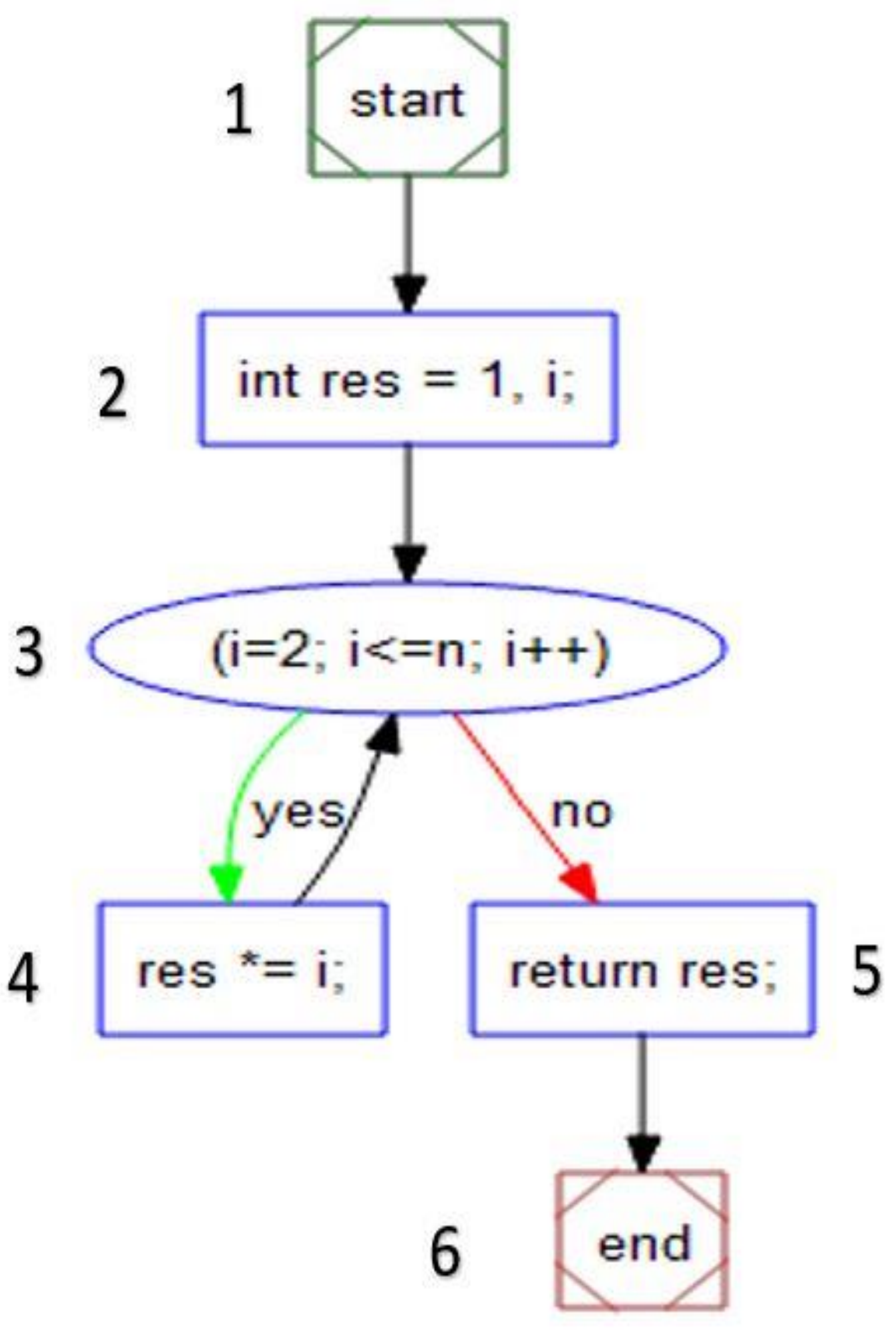


Figure 2: Factorial CFG

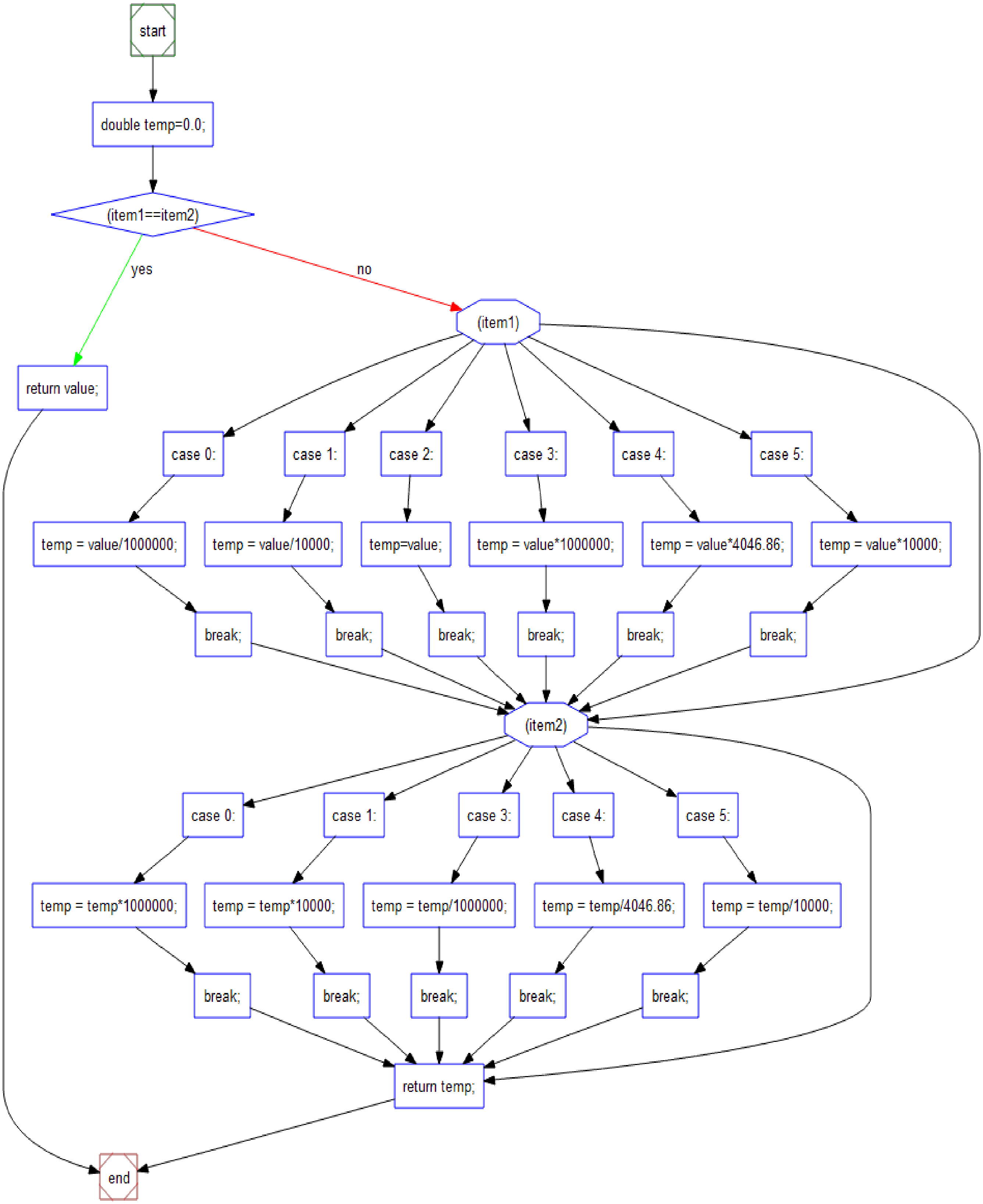


Figure 3: Area Conversion CFG

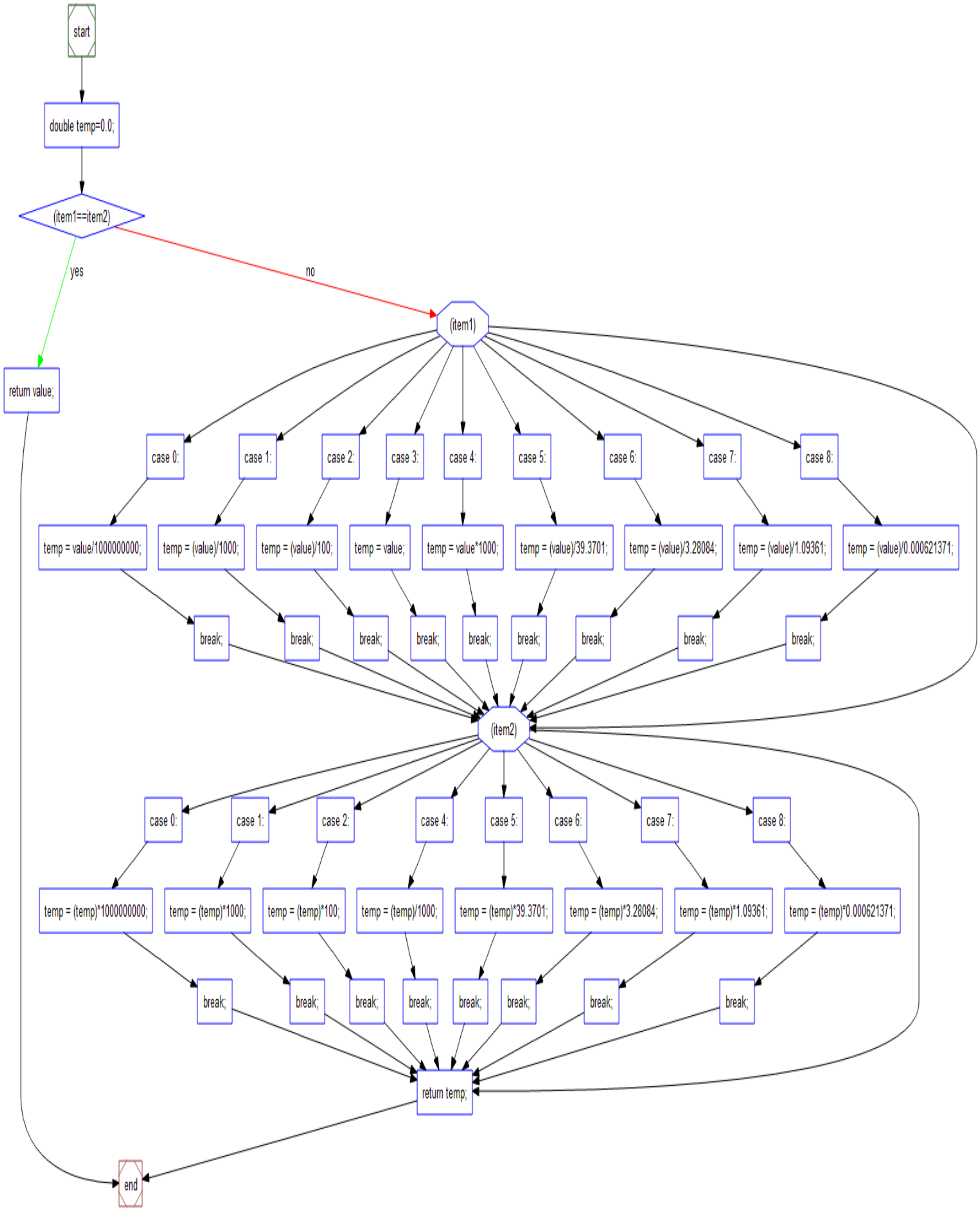


Figure 4: Length Conversion CFG

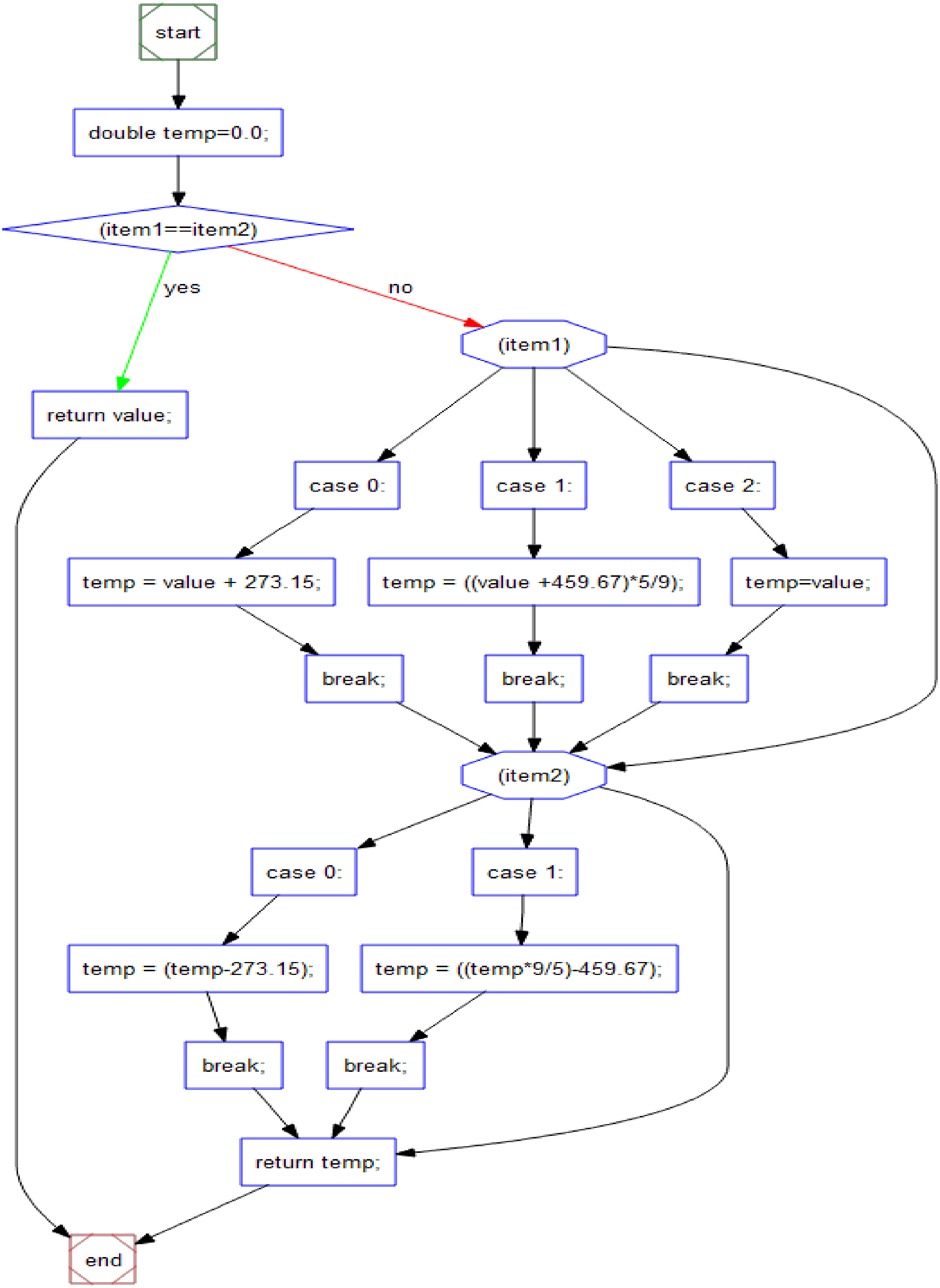


Figure 5: Temperature Conversion CFG

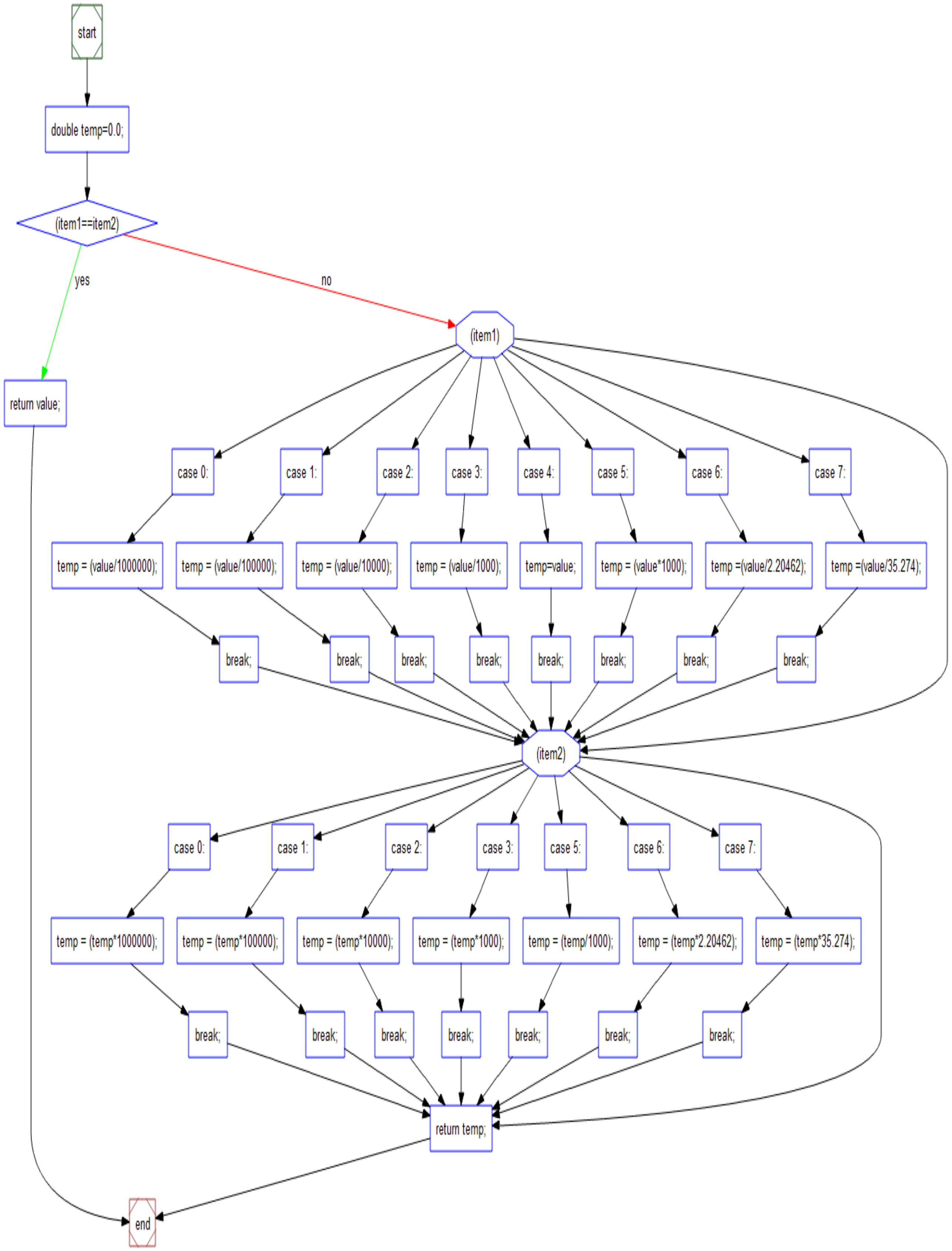


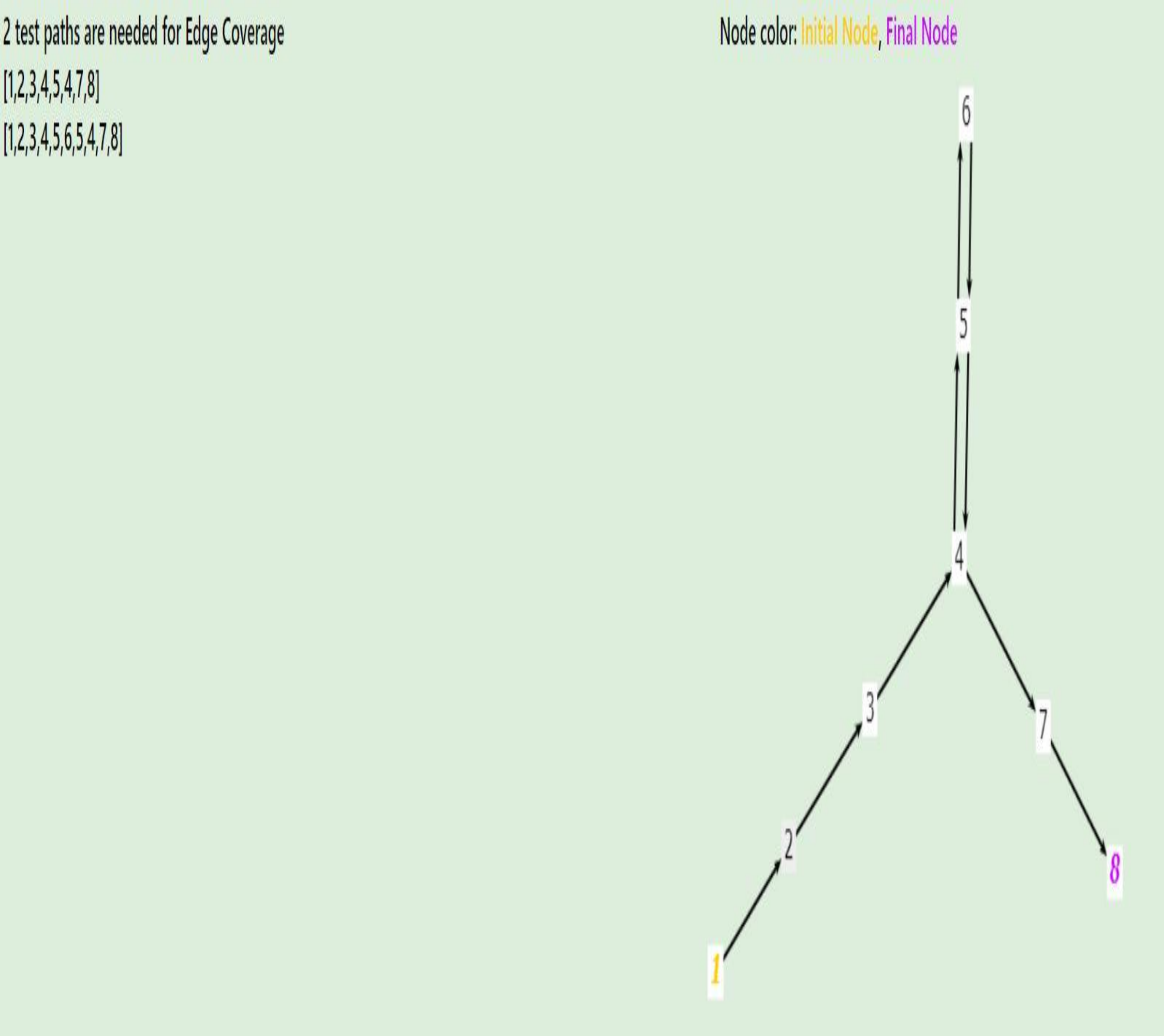
Figure 6: Weight Conversion CFG

Edge Coverage and Prime Path Coverage

Tools Used:
Android Studio (https://developer.android.com/studio),
JUnit5 (https://junit.org/junit5/),
George Masen University Tool
(https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage)

Combination Feature Testing

Test Requirement for Edge Coverage



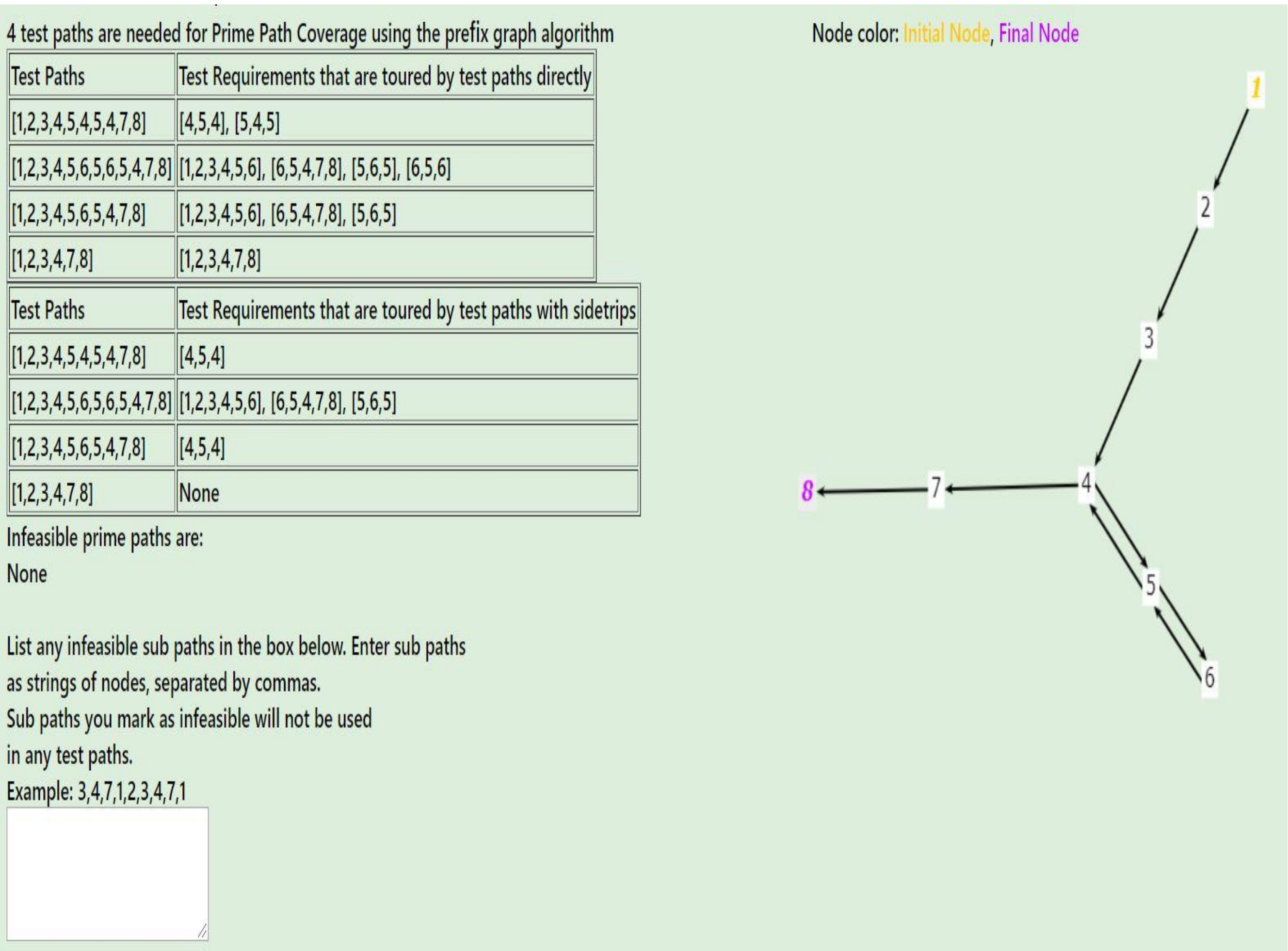
2 test paths are needed for Edge Coverage

[1,2,3,4,5,4,7,8]

[1,2,3,4,5,6,5,4,7,8]

Node color: Initial Node, Final Node

Test Requirement for Prime Path Coverage



4 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Node color: Initial Node, Final Node

Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,4,5,4,5,4,7,8]	[4,5,4], [5,4,5]
[1,2,3,4,5,6,5,6,5,4,7,8]	[1,2,3,4,5,6], [6,5,4,7,8], [5,6,5], [6,5,6]
[1,2,3,4,5,6,5,4,7,8]	[1,2,3,4,5,6], [6,5,4,7,8], [5,6,5]
[1,2,3,4,7,8]	[1,2,3,4,7,8]

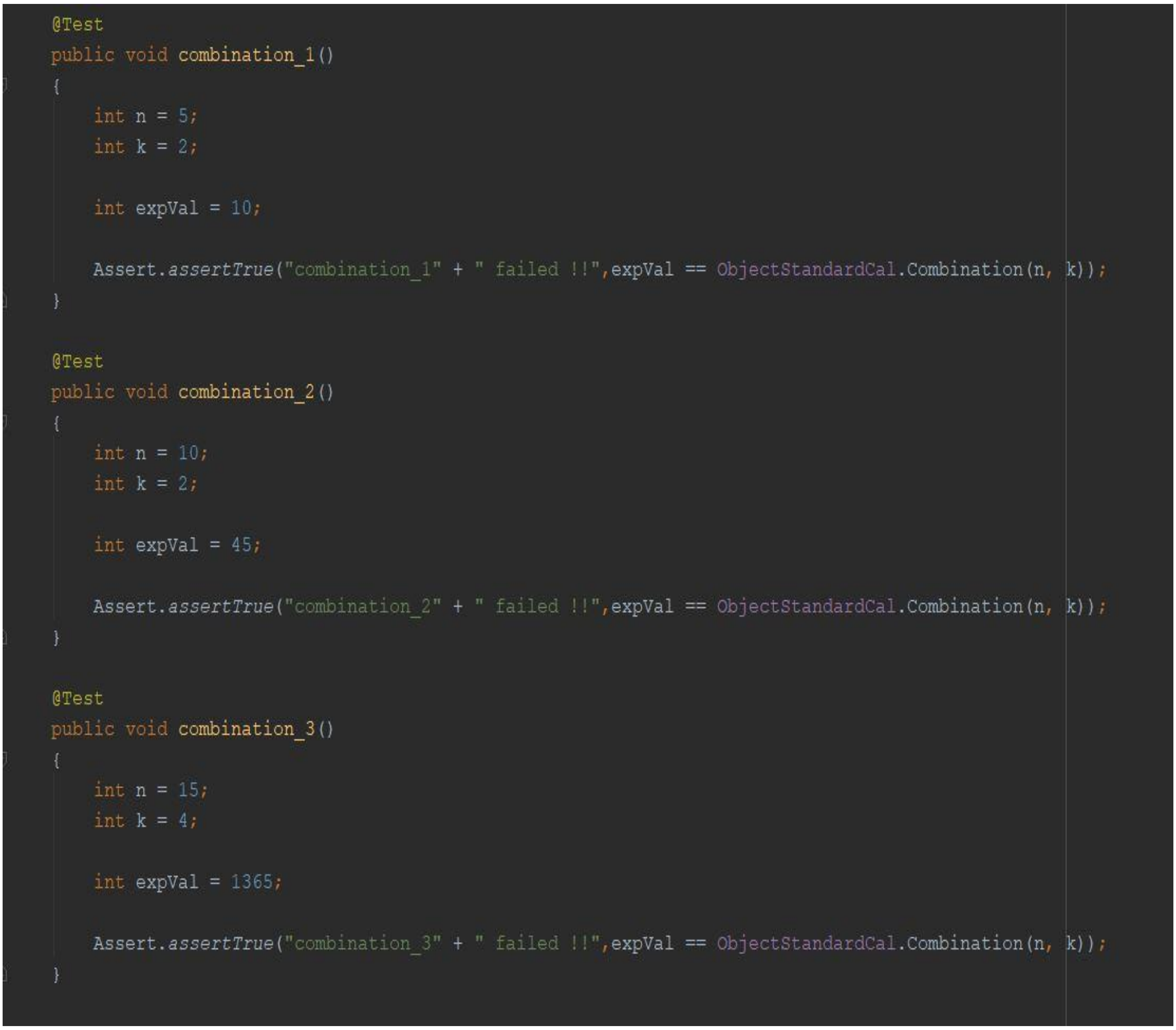
Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,3,4,5,4,5,4,7,8]	[4,5,4]
[1,2,3,4,5,6,5,6,5,4,7,8]	[1,2,3,4,5,6], [6,5,4,7,8], [5,6,5]
[1,2,3,4,5,6,5,4,7,8]	[4,5,4]
[1,2,3,4,7,8]	None

Infeasible prime paths are:
None

List any infeasible sub paths in the box below. Enter sub paths as strings of nodes, separated by commas. Sub paths you mark as infeasible will not be used in any test paths.
Example: 3,4,7,1,2,3,4,7,1

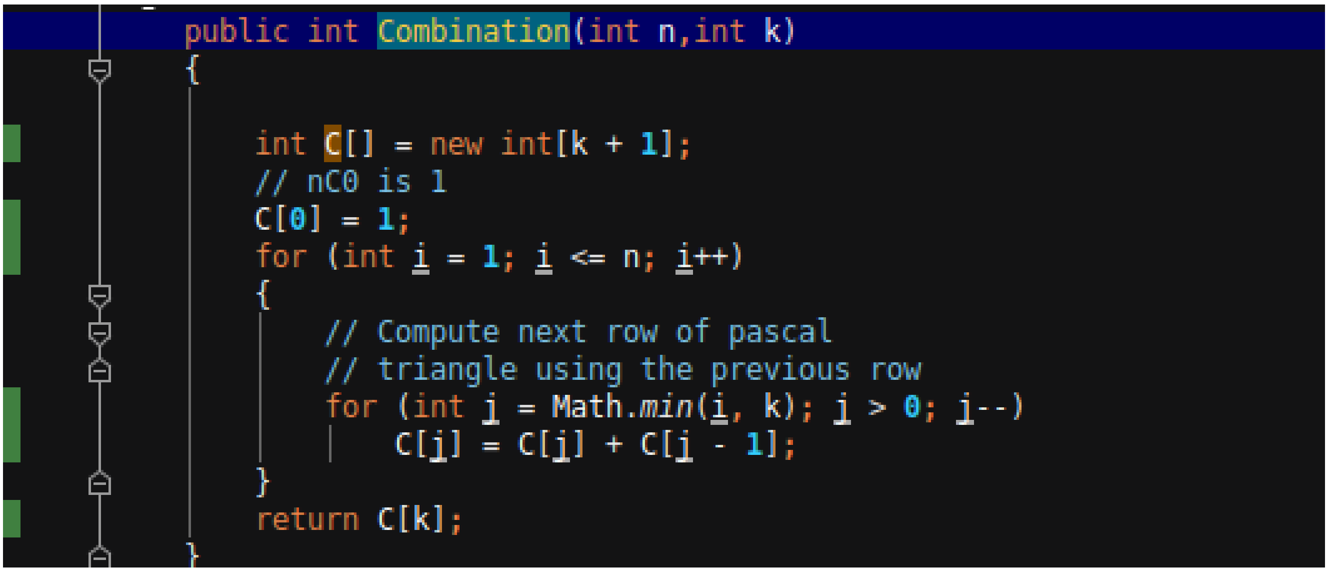
Test Cases

S.no	N	K	Expected Output	Actual Output	Status
1	5	2	10	10	Pass
2	10	2	45	45	Pass
3	15	4	1365	1365	Pass
4	20	10	184756	184756	Pass
5	5	1	10	5	Fail (Negative Test Case)



```java
@Test
public void combination_1()
{
    int n = 5;
    int k = 2;

    int expVal = 10;

    Assert.assertTrue("combination_1" + " failed !!",expVal == ObjectStandardCal.Combination(n, k));
}

@Test
public void combination_2()
{
    int n = 10;
    int k = 2;

    int expVal = 45;

    Assert.assertTrue("combination_2" + " failed !!",expVal == ObjectStandardCal.Combination(n, k));
}

@Test
public void combination_3()
{
    int n = 15;
    int k = 4;

    int expVal = 1365;

    Assert.assertTrue("combination_3" + " failed !!",expVal == ObjectStandardCal.Combination(n, k));
}
```

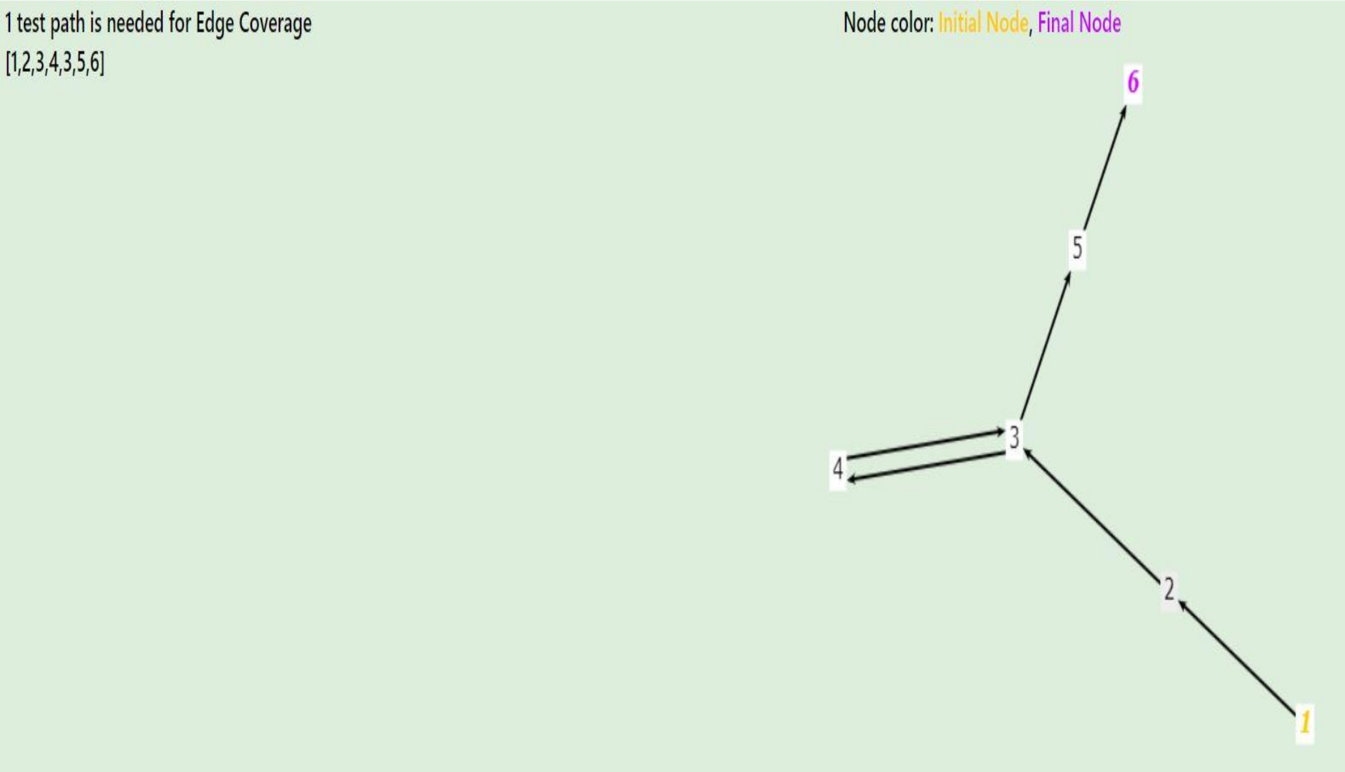
Code Coverage



```java
public int Combination(int n,int k)
{

    int C[] = new int[k + 1];
    // nC0 is 1
    C[0] = 1;
    for (int i = 1; i <= n; i++)
    {
        // Compute next row of pascal
        // triangle using the previous row
        for (int j = Math.min(i, k); j > 0; j--)
            C[j] = C[j] + C[j - 1];
    }
    return C[k];
}
```

Factorial Feature Testing

Test Requirement for Edge Coverage

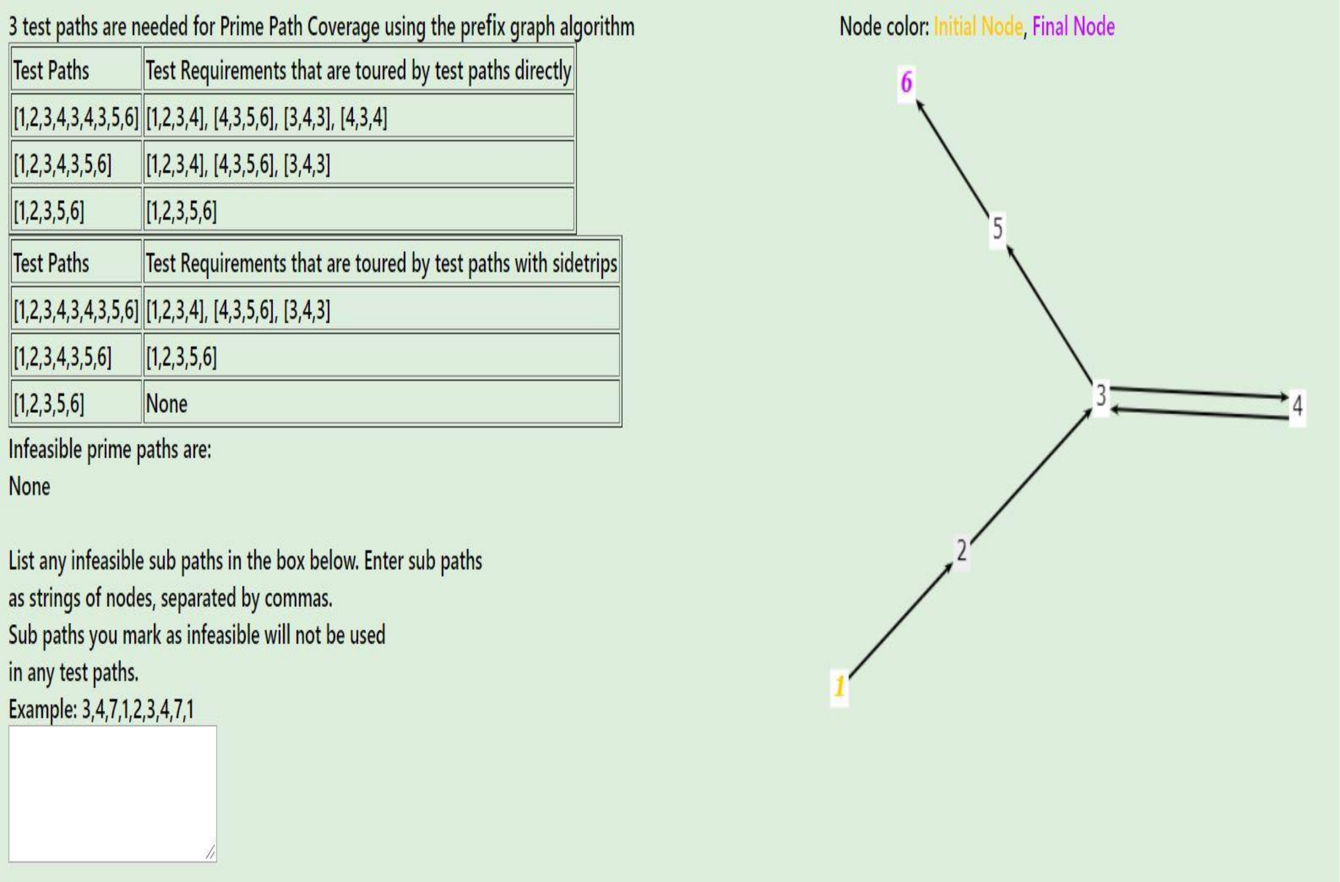


1 test path is needed for Edge Coverage

[1,2,3,4,3,5,6]

Node color: Initial Node, Final Node

Test Requirement for Prime Path Coverage



3 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,4,3,4,3,5,6]	[1,2,3,4], [4,3,5,6], [3,4,3], [4,3,4]
[1,2,3,4,3,5,6]	[1,2,3,4], [4,3,5,6], [3,4,3]
[1,2,3,5,6]	[1,2,3,5,6]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,3,4,3,4,3,5,6]	[1,2,3,4], [4,3,5,6], [3,4,3]
[1,2,3,4,3,5,6]	[1,2,3,5,6]
[1,2,3,5,6]	None

Infeasible prime paths are:

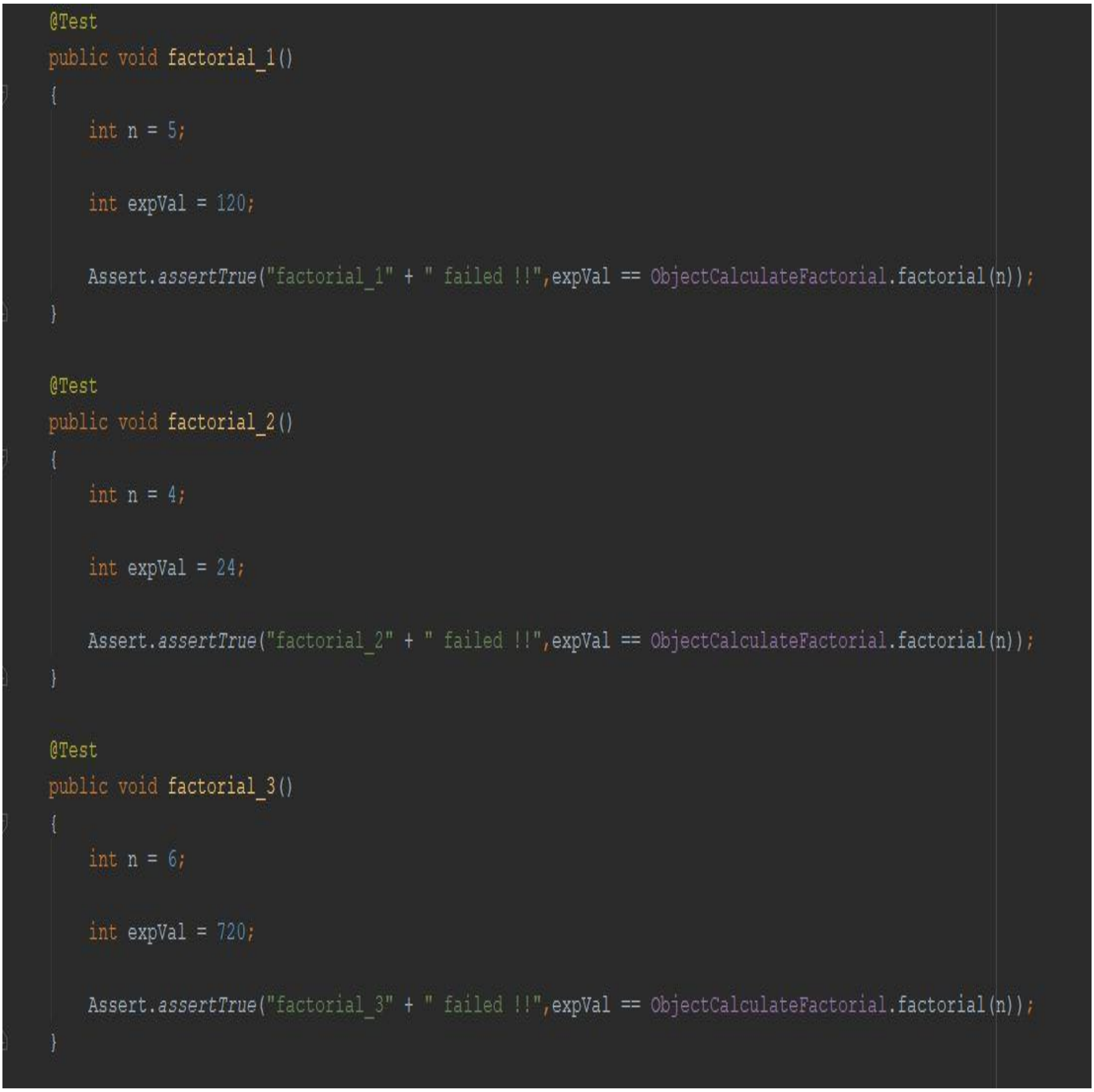
None

List any infeasible sub paths in the box below. Enter sub paths as strings of nodes, separated by commas.
Sub paths you mark as infeasible will not be used in any test paths.
Example: 3,4,7,1,2,3,4,7,1

Node color: Initial Node, Final Node

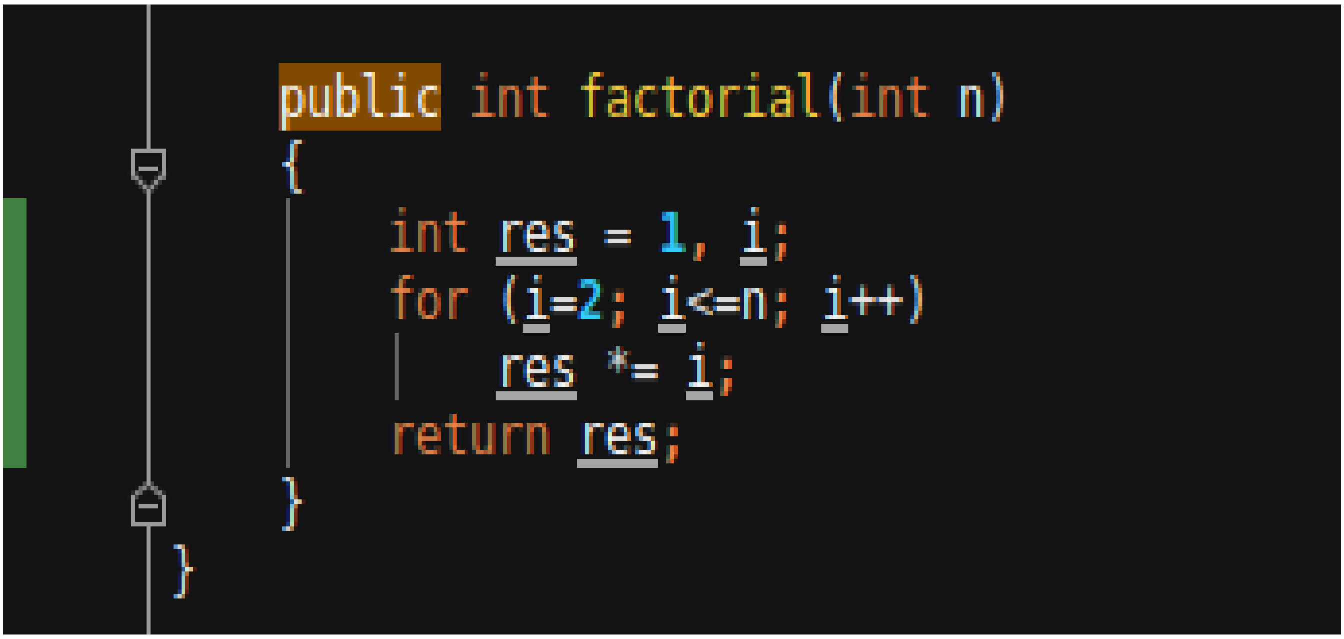
Test Cases

S.no	N	Expected Output	Actual Output	Status
1	5	120	120	Pass
2	4	24	24	Pass
3	6	720	720	Pass
4	3	6	6	Pass
5	5	120	20	Fail



```
@Test
public void factorial_1()
{
    int n = 5;

    int expVal = 120;

    Assert.assertTrue("factorial_1" + " failed !!",expVal == ObjectCalculateFactorial.factorial(n));
}

@Test
public void factorial_2()
{
    int n = 4;

    int expVal = 24;

    Assert.assertTrue("factorial_2" + " failed !!",expVal == ObjectCalculateFactorial.factorial(n));
}

@Test
public void factorial_3()
{
    int n = 6;

    int expVal = 720;

    Assert.assertTrue("factorial_3" + " failed !!",expVal == ObjectCalculateFactorial.factorial(n));
}
```

Code Coverage

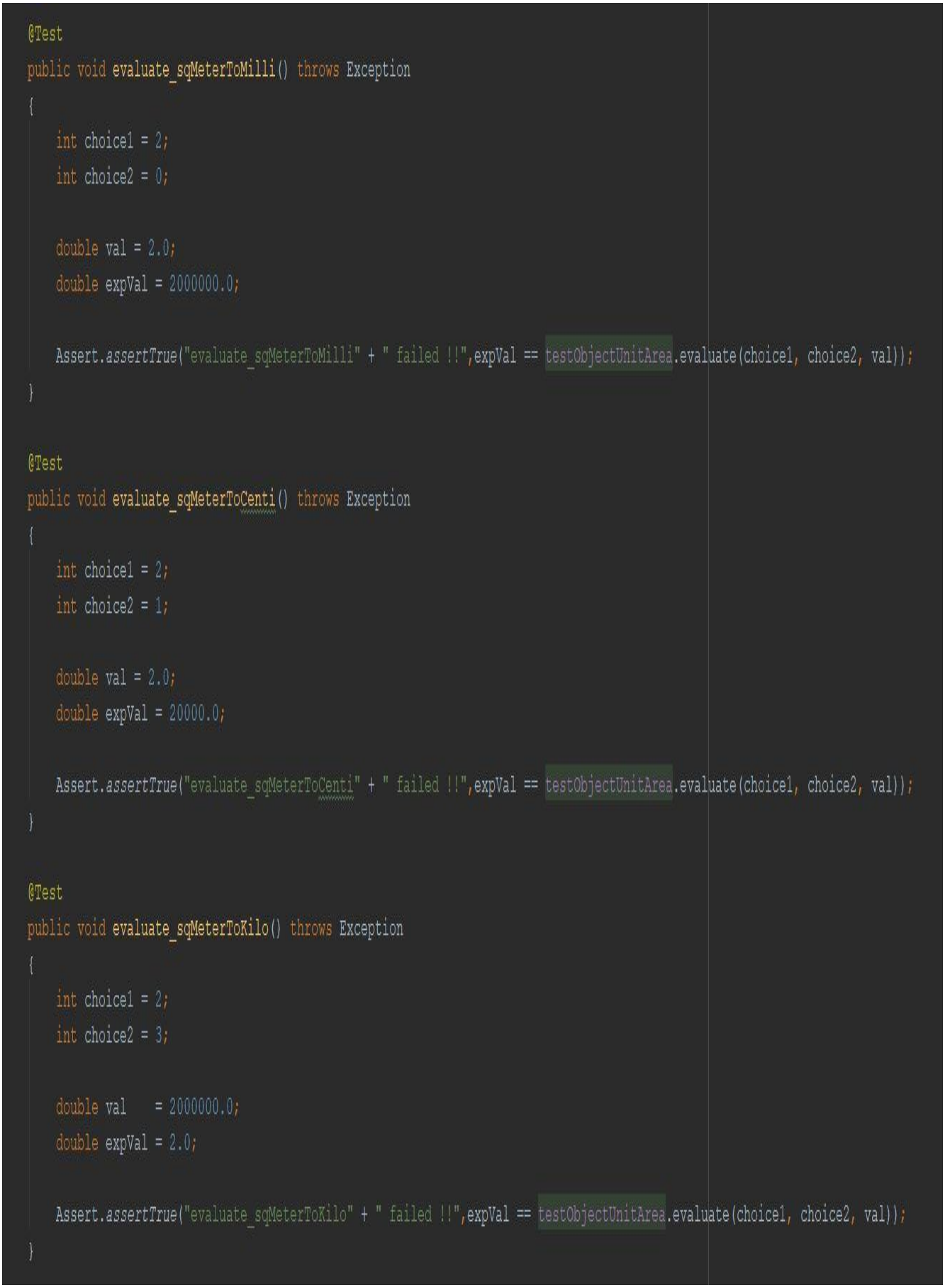


```
public int factorial(int n)
{
    int res = 1, i;
    for (i=2; i<=n; i++)
        res *= i;
    return res;
}
}
```

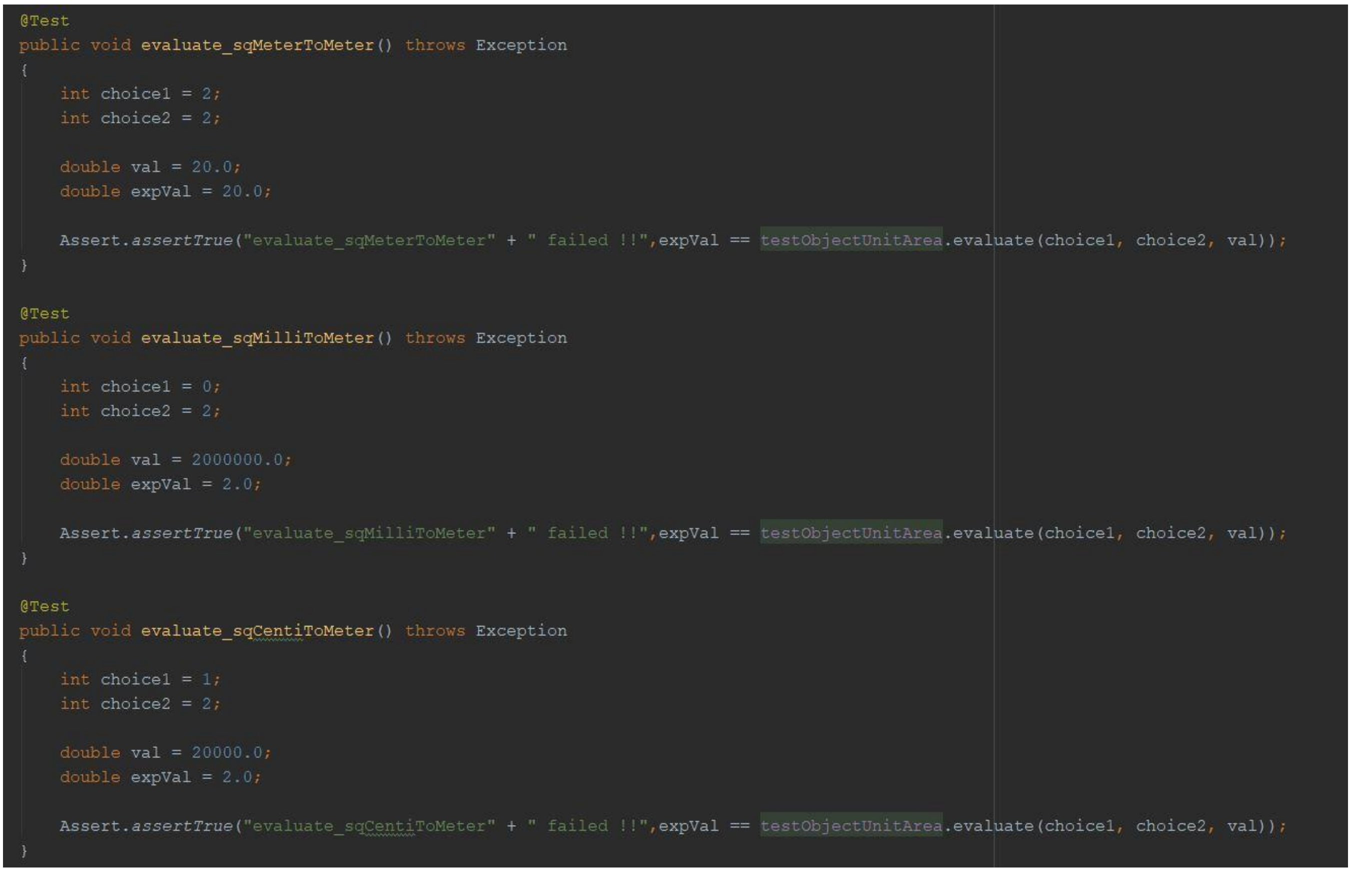
Area Unit Converter Feature Testing

Test Cases

S.no	Choice1	Choice 2	Value	Expected Output	Actual Output	Status
1	2	2	20	20	20	Pass
2	0	2	2000000	2	2	Pass
3	1	2	20000	2	2	Pass
4	3	2	2	2000000	2000000	Pass
5	4	2	1	4046.86	4046.86	Pass
6	5	2	1	10000	10000	Pass
7	2	0	2	2000000	2000000	Pass
8	2	1	2	20000	20000	Pass
9	2	3	2000000	2	2	Pass
10	2	4	4046.86	1	1	Pass
11	2	5	20	20	.0002	Fail (Negative Test case)

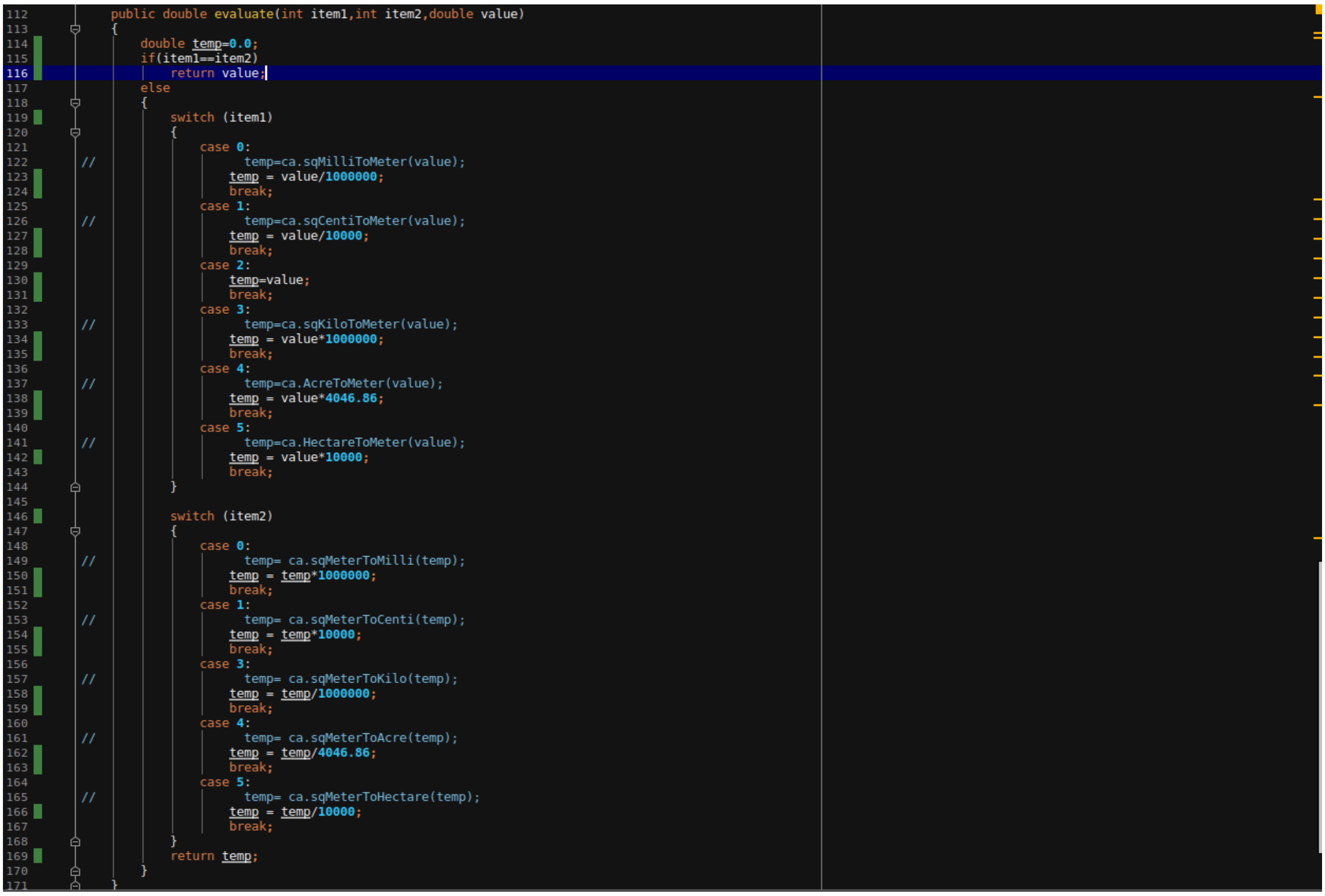


```
@Test
public void evaluate_sqMeterToMilli() throws Exception
{
    int choice1 = 2;
    int choice2 = 0;

    double val = 2.0;
    double expVal = 2000000.0;

    Assert.assertTrue("evaluate_sqMeterToMilli" + " failed !!",expVal == testObjectUnitArea.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_sqMeterToCenti() throws Exception
{
    int choice1 = 2;
    int choice2 = 1;

    double val = 2.0;
    double expVal = 20000.0;

    Assert.assertTrue("evaluate_sqMeterToCenti" + " failed !!",expVal == testObjectUnitArea.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_sqMeterToKilo() throws Exception
{
    int choice1 = 2;
    int choice2 = 3;

    double val    = 2000000.0;
    double expVal = 2.0;

    Assert.assertTrue("evaluate_sqMeterToKilo" + " failed !!",expVal == testObjectUnitArea.evaluate(choice1, choice2, val));
}
```



```java
@Test
public void evaluate_sqMeterToMeter() throws Exception
{
    int choice1 = 2;
    int choice2 = 2;

    double val = 20.0;
    double expVal = 20.0;

    Assert.assertTrue("evaluate_sqMeterToMeter" + " failed !!",expVal == testObjectUnitArea.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_sqMilliToMeter() throws Exception
{
    int choice1 = 0;
    int choice2 = 2;

    double val = 2000000.0;
    double expVal = 2.0;

    Assert.assertTrue("evaluate_sqMilliToMeter" + " failed !!",expVal == testObjectUnitArea.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_sqCentiToMeter() throws Exception
{
    int choice1 = 1;
    int choice2 = 2;

    double val = 20000.0;
    double expVal = 2.0;

    Assert.assertTrue("evaluate_sqCentiToMeter" + " failed !!",expVal == testObjectUnitArea.evaluate(choice1, choice2, val));
}
```

Code Coverage

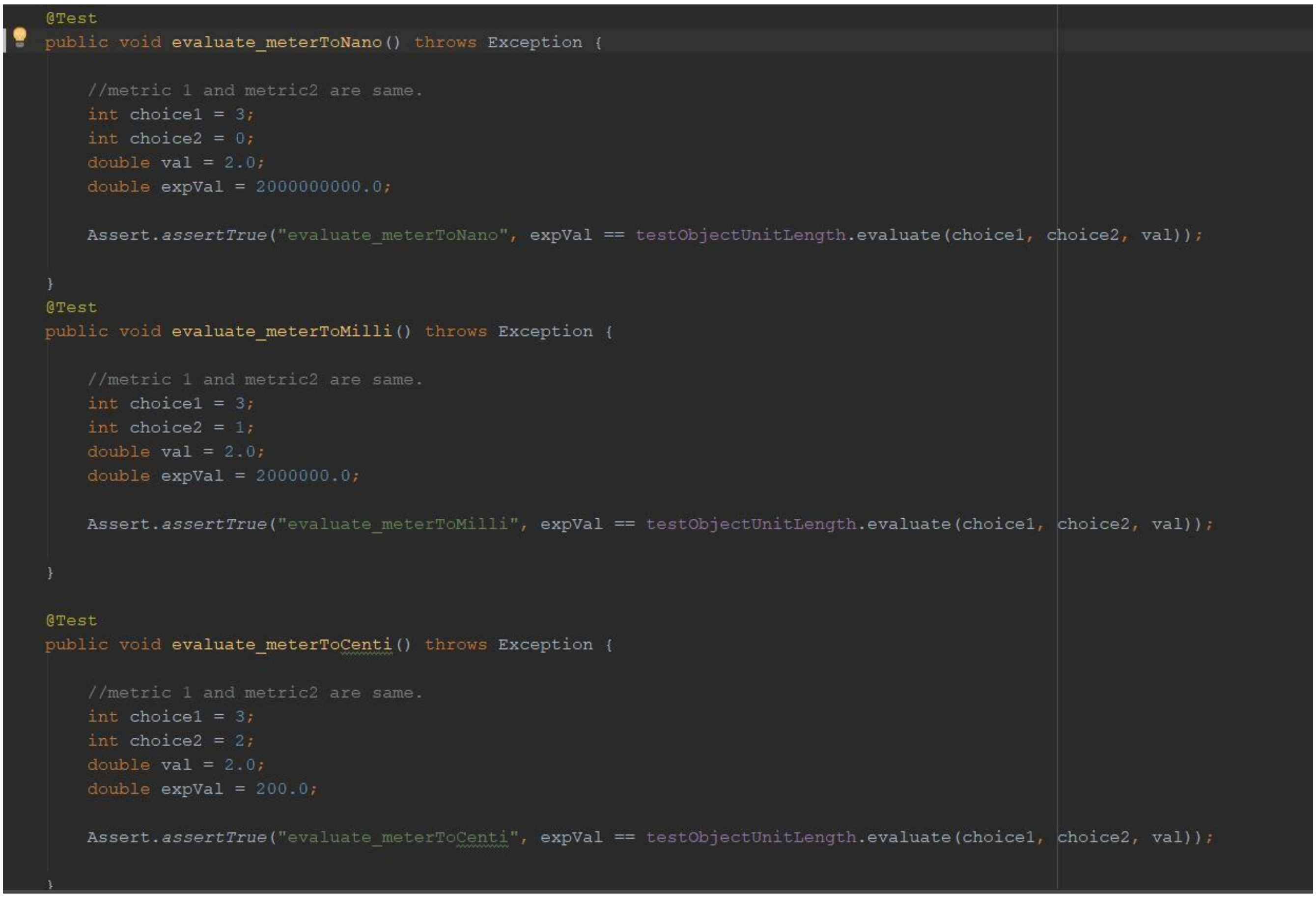


```java
public double evaluate(int item1,int item2,double value)
{
    double temp=0.0;
    if(item1==item2)
        return value;
    else
    {
        switch (item1)
        {
            case 0:
//                temp=ca.sqMilliToMeter(value);
                temp = value/1000000;
                break;
            case 1:
//                temp=ca.sqCentiToMeter(value);
                temp = value/10000;
                break;
            case 2:
                temp=value;
                break;
            case 3:
//                temp=ca.sqKiloToMeter(value);
                temp = value*1000000;
                break;
            case 4:
//                temp=ca.AcreToMeter(value);
                temp = value*4046.86;
                break;
            case 5:
//                temp=ca.HectareToMeter(value);
                temp = value*10000;
                break;
        }

        switch (item2)
        {
            case 0:
//                temp= ca.sqMeterToMilli(temp);
                temp = temp*1000000;
                break;
            case 1:
//                temp= ca.sqMeterToCenti(temp);
                temp = temp*10000;
                break;
            case 3:
//                temp= ca.sqMeterToKilo(temp);
                temp = temp/1000000;
                break;
            case 4:
//                temp= ca.sqMeterToAcre(temp);
                temp = temp/4046.86;
                break;
            case 5:
//                temp= ca.sqMeterToHectare(temp);
                temp = temp/10000;
                break;
        }
        return temp;
    }
}
```

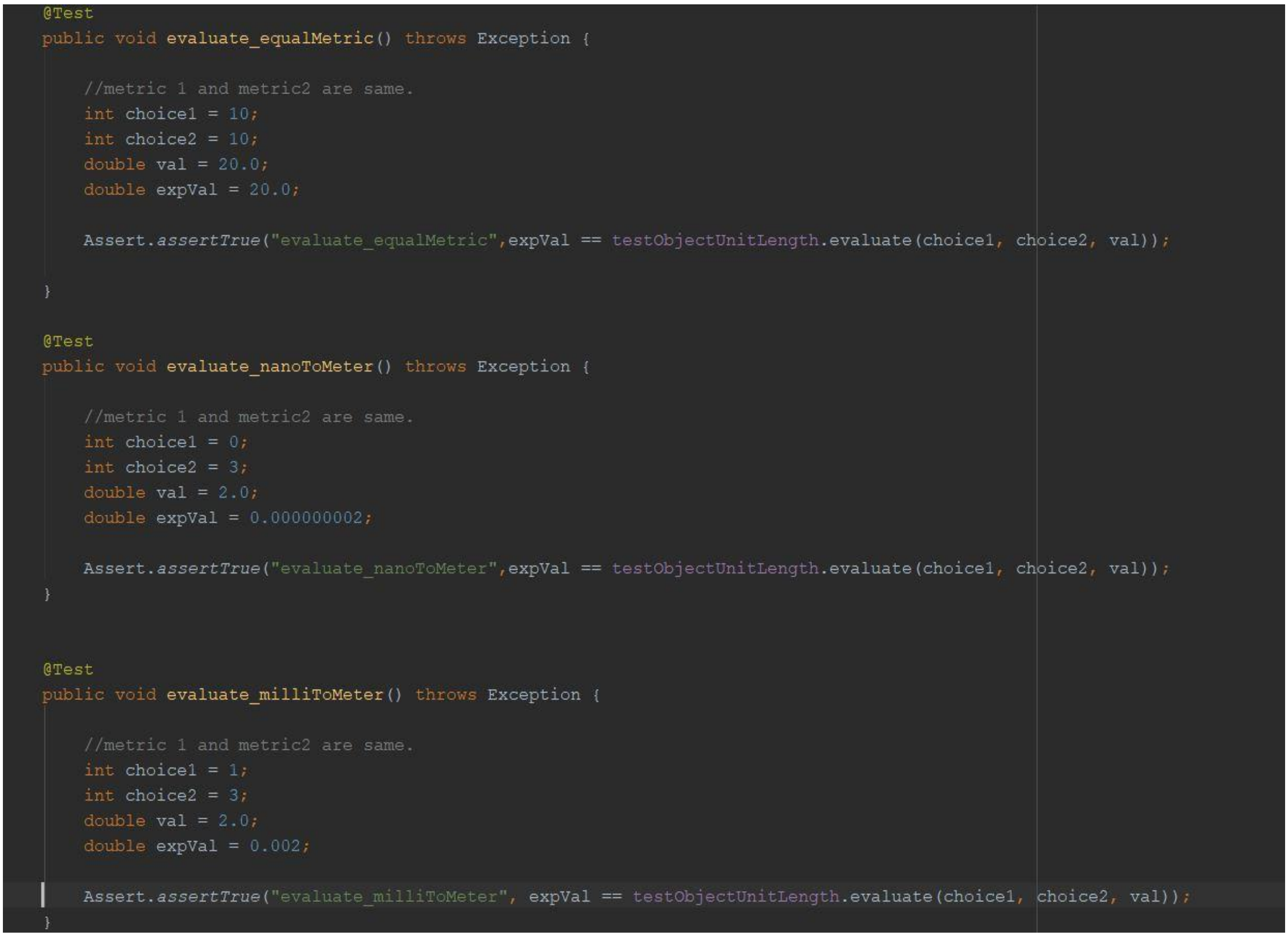
Length Unit Converter Testing

Test Cases

S.no	Choice1	Choice2	Value	Expected Output	Actual Output	Status
1	5	5	20	20	20	Pass
2	0	3	2	2E-9	2E-9	Pass
3	1	3	2	.002	.002	Pass
4	2	3	2	.02	.002	Pass
5	3	3	2	2	2	Pass
6	4	3	2	2000	2000	Pass
7	5	3	2	78.7402	.0508	Fail (Negative TestCase)
8	6	3	2	6.56168	0.6096	Fail (Negative TestCase)
9	7	3	2	1.8287988	1.82	Fail (Negative TestCase)
10	8	3	2	3281.68895	3218.69	Fail (Negative TestCase)
11	3	0	2	2E+9	2E+9	Pass
12	3	1	2	2E+6	2E+3	Fail (Negative TestCase)
13	3	2	2	200	200	Pass
14	3	4	2	2E-3	2E-3	Pass
15	3	5	2	78.7402	78.7402	Pass
16	3	6	2	6.56168	6.56168	Pass
17	3	7	2	2.18722	2.18722	Pass
18	3	8	2	1242742E-6	1242742E-6	Pass

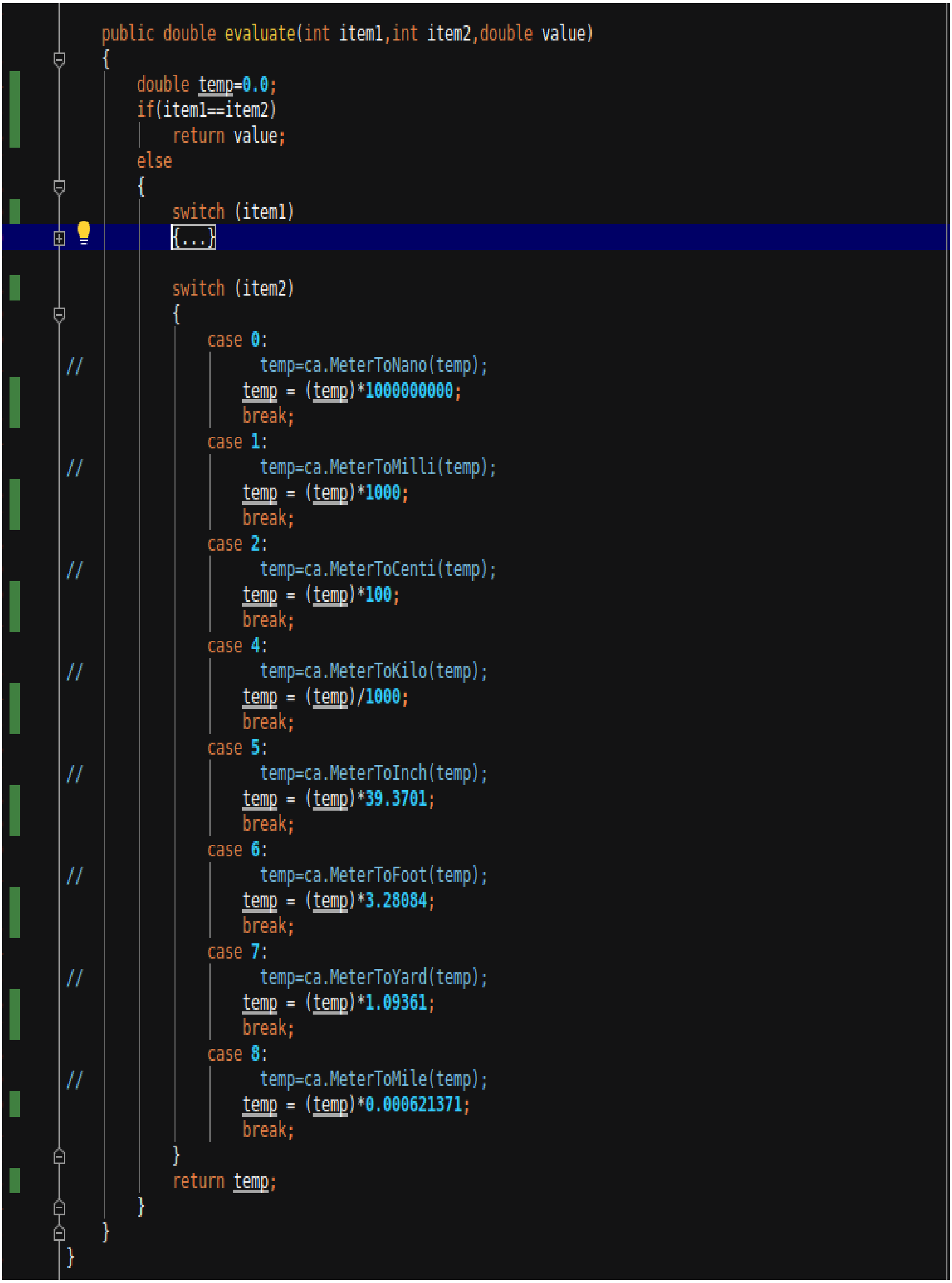


```
@Test
public void evaluate_meterToNano() throws Exception {

    //metric 1 and metric2 are same.
    int choice1 = 3;
    int choice2 = 0;
    double val = 2.0;
    double expVal = 2000000000.0;

    Assert.assertTrue("evaluate_meterToNano", expVal == testObjectUnitLength.evaluate(choice1, choice2, val));

}
@Test
public void evaluate_meterToMilli() throws Exception {

    //metric 1 and metric2 are same.
    int choice1 = 3;
    int choice2 = 1;
    double val = 2.0;
    double expVal = 2000000.0;

    Assert.assertTrue("evaluate_meterToMilli", expVal == testObjectUnitLength.evaluate(choice1, choice2, val));

}

@Test
public void evaluate_meterToCenti() throws Exception {

    //metric 1 and metric2 are same.
    int choice1 = 3;
    int choice2 = 2;
    double val = 2.0;
    double expVal = 200.0;

    Assert.assertTrue("evaluate_meterToCenti", expVal == testObjectUnitLength.evaluate(choice1, choice2, val));

}
```



```
@Test
public void evaluate_equalMetric() throws Exception {

    //metric 1 and metric2 are same.
    int choice1 = 10;
    int choice2 = 10;
    double val = 20.0;
    double expVal = 20.0;

    Assert.assertTrue("evaluate_equalMetric",expVal == testObjectUnitLength.evaluate(choice1, choice2, val));

}

@Test
public void evaluate_nanoToMeter() throws Exception {

    //metric 1 and metric2 are same.
    int choice1 = 0;
    int choice2 = 3;
    double val = 2.0;
    double expVal = 0.000000002;

    Assert.assertTrue("evaluate_nanoToMeter",expVal == testObjectUnitLength.evaluate(choice1, choice2, val));
}


@Test
public void evaluate_milliToMeter() throws Exception {

    //metric 1 and metric2 are same.
    int choice1 = 1;
    int choice2 = 3;
    double val = 2.0;
    double expVal = 0.002;

    Assert.assertTrue("evaluate_milliToMeter", expVal == testObjectUnitLength.evaluate(choice1, choice2, val));
}
```

Code Coverage

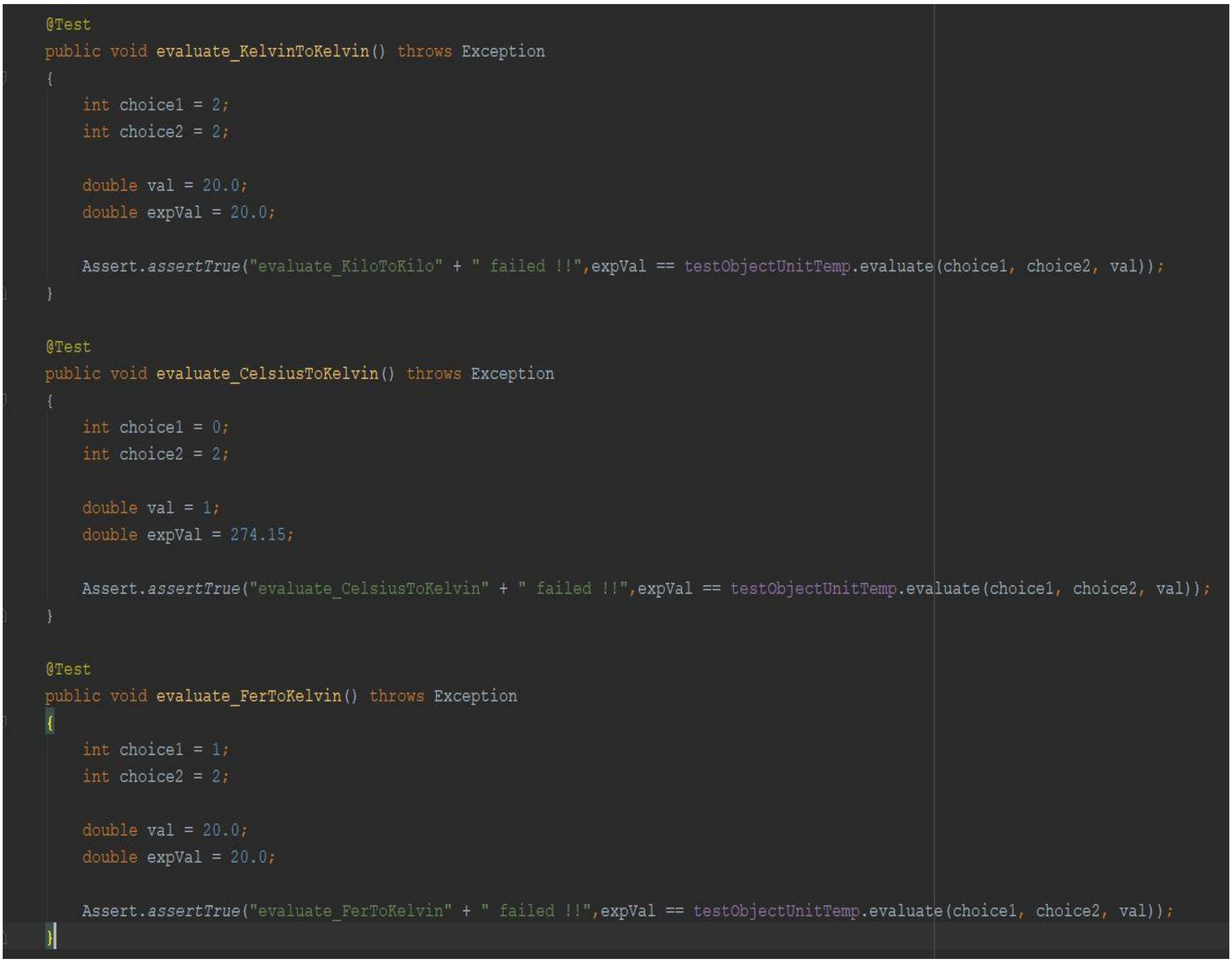


```
    public double evaluate(int item1,int item2,double value)
    {
        double temp=0.0;
        if(item1==item2)
            return value;
        else
        {
            switch (item1)
            {...}

            switch (item2)
            {
                case 0:
//                    temp=ca.MeterToNano(temp);
                    temp = (temp)*1000000000;
                    break;
                case 1:
//                    temp=ca.MeterToMilli(temp);
                    temp = (temp)*1000;
                    break;
                case 2:
//                    temp=ca.MeterToCenti(temp);
                    temp = (temp)*100;
                    break;
                case 4:
//                    temp=ca.MeterToKilo(temp);
                    temp = (temp)/1000;
                    break;
                case 5:
//                    temp=ca.MeterToInch(temp);
                    temp = (temp)*39.3701;
                    break;
                case 6:
//                    temp=ca.MeterToFoot(temp);
                    temp = (temp)*3.28084;
                    break;
                case 7:
//                    temp=ca.MeterToYard(temp);
                    temp = (temp)*1.09361;
                    break;
                case 8:
//                    temp=ca.MeterToMile(temp);
                    temp = (temp)*0.000621371;
                    break;
            }
            return temp;
        }
    }
}
```

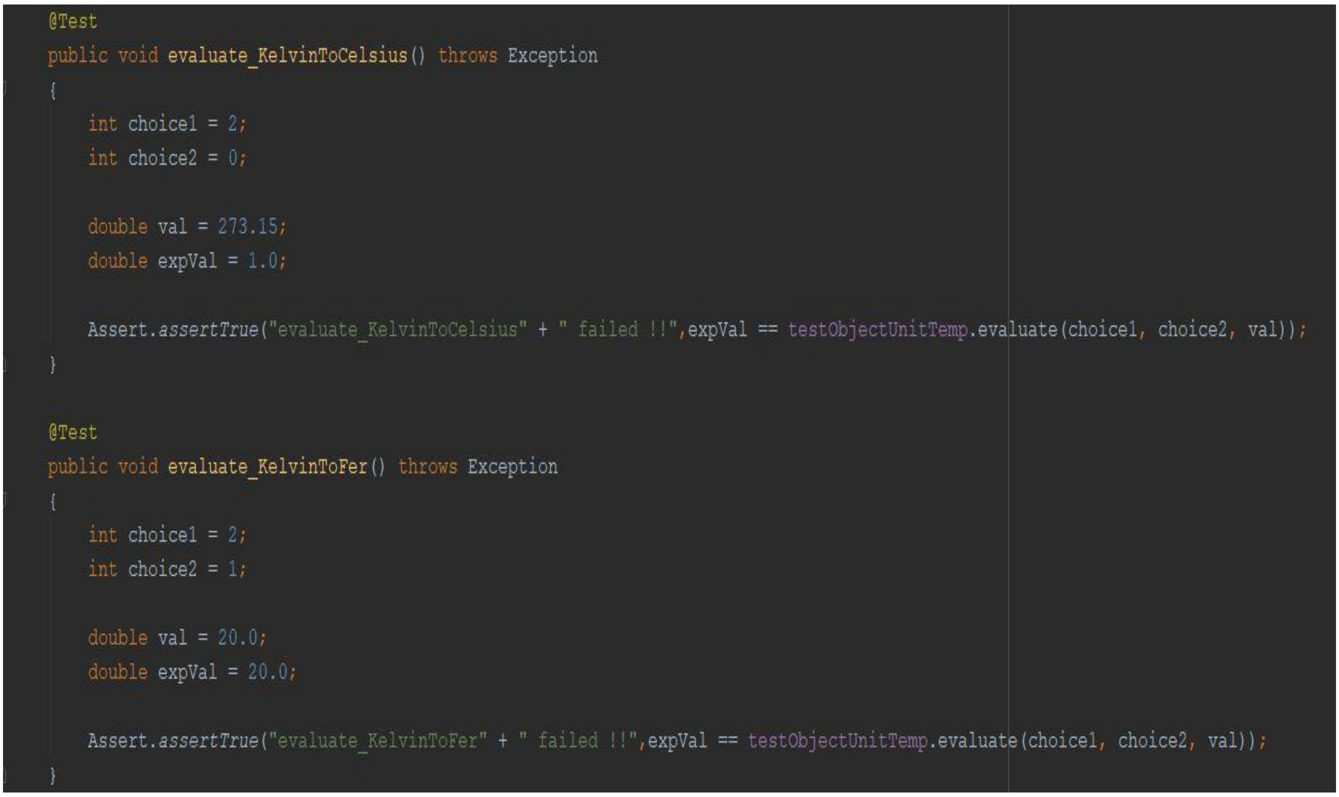
Temperature Unit Converter Testing

Test Cases

S.no	Choice1	Choice2	Value	Expected Output	Actual Output	Status
1	2	2	20	20	20	Pass
2	0	2	1	274.15	274.15	Pass
3	1	2	20	20	266.483	Fail (Negative Test Case)
4	2	0	274.15	1	1	Pass
5	2	1	20	20	-423.67	Fail (Negative Test Case)

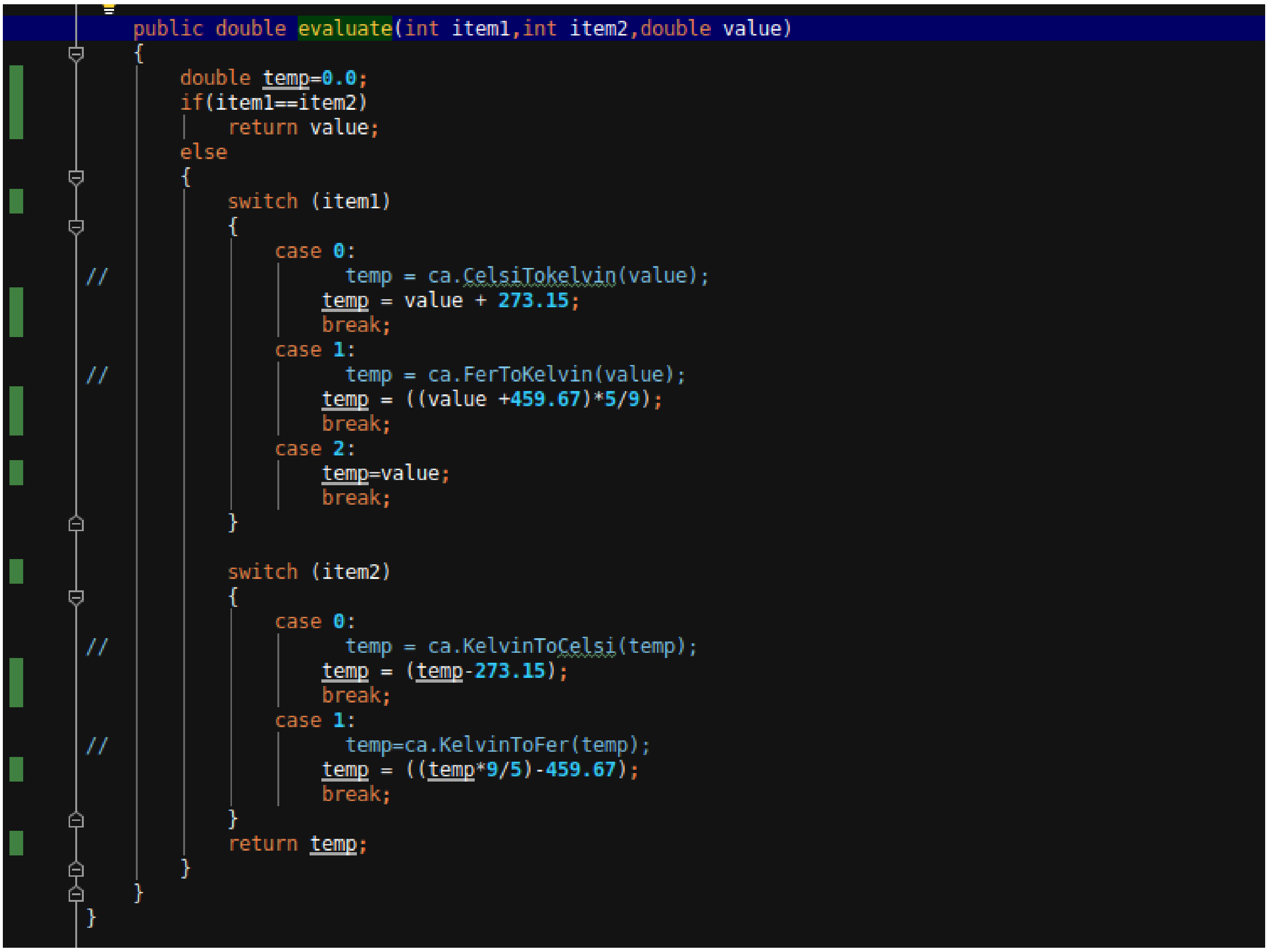


```java
@Test
public void evaluate_KelvinToKelvin() throws Exception
{
    int choice1 = 2;
    int choice2 = 2;

    double val = 20.0;
    double expVal = 20.0;

    Assert.assertTrue("evaluate_KiloToKilo" + " failed !!",expVal == testObjectUnitTemp.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_CelsiusToKelvin() throws Exception
{
    int choice1 = 0;
    int choice2 = 2;

    double val = 1;
    double expVal = 274.15;

    Assert.assertTrue("evaluate_CelsiusToKelvin" + " failed !!",expVal == testObjectUnitTemp.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_FerToKelvin() throws Exception
{
    int choice1 = 1;
    int choice2 = 2;

    double val = 20.0;
    double expVal = 20.0;

    Assert.assertTrue("evaluate_FerToKelvin" + " failed !!",expVal == testObjectUnitTemp.evaluate(choice1, choice2, val));
}
```



```
@Test
public void evaluate_KelvinToCelsius() throws Exception
{
    int choice1 = 2;
    int choice2 = 0;

    double val = 273.15;
    double expVal = 1.0;

    Assert.assertTrue("evaluate_KelvinToCelsius" + " failed !!",expVal == testObjectUnitTemp.evaluate(choice1, choice2, val));
}

@Test
public void evaluate_KelvinToFer() throws Exception
{
    int choice1 = 2;
    int choice2 = 1;

    double val = 20.0;
    double expVal = 20.0;

    Assert.assertTrue("evaluate_KelvinToFer" + " failed !!",expVal == testObjectUnitTemp.evaluate(choice1, choice2, val));
}
```

Code Coverage

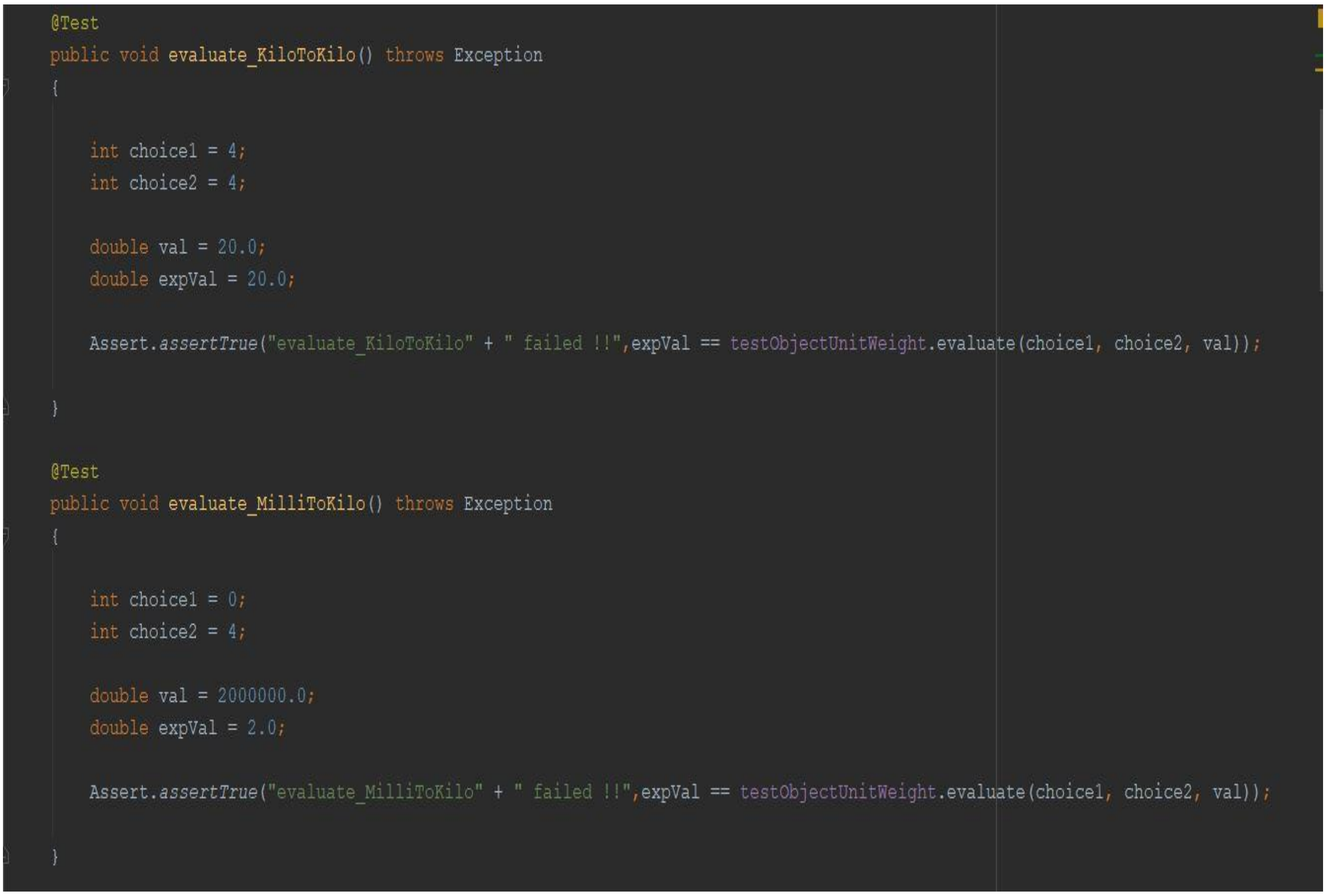


```
    public double evaluate(int item1,int item2,double value)
    {
        double temp=0.0;
        if(item1==item2)
            return value;
        else
        {
            switch (item1)
            {
                case 0:
//                      temp = ca.CelsiTokelvin(value);
                    temp = value + 273.15;
                    break;
                case 1:
//                      temp = ca.FerToKelvin(value);
                    temp = ((value +459.67)*5/9);
                    break;
                case 2:
                    temp=value;
                    break;
            }

            switch (item2)
            {
                case 0:
//                      temp = ca.KelvinToCelsi(temp);
                    temp = (temp-273.15);
                    break;
                case 1:
//                      temp=ca.KelvinToFer(temp);
                    temp = ((temp*9/5)-459.67);
                    break;
            }
            return temp;
        }
    }
}
```

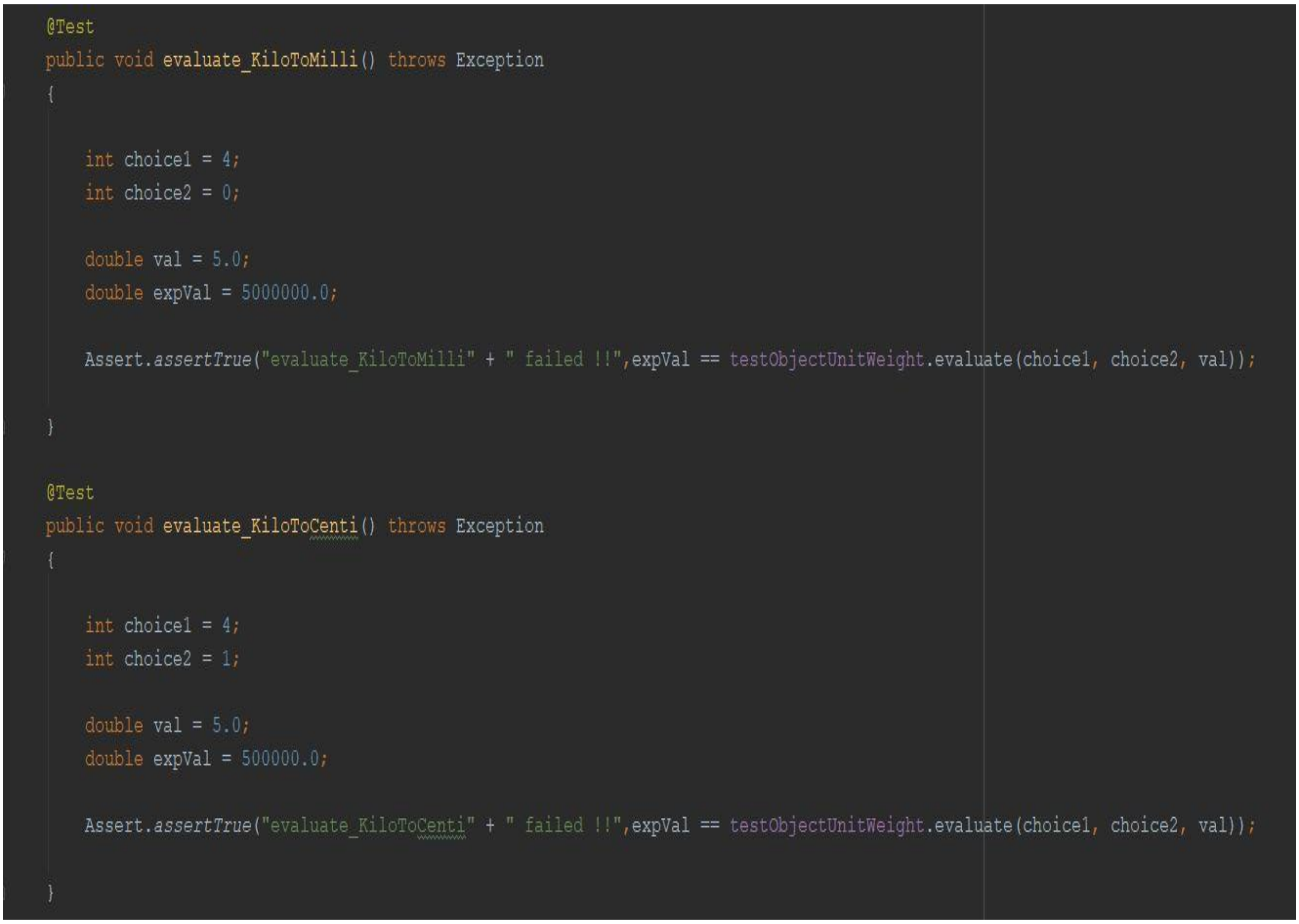
Weight Unit Converter Testing

Test Cases

S.no	Choice1	Choice2	Value	Expected Output	Actual Output	Status
1	4	4	20	20	20	Pass
2	0	4	2000000	2	2	Pass
3	1	4	200000	2	2	Pass
4	2	4	20000	2	2	Pass
5	3	4	200	2	.2	Fail (Negative TestCase)
6	5	4	2	2000	2000	Pass
7	6	4	2.20462	1	1	Pass
8	7	4	70	1	1.98447	Fail (Negative TestCase)
9	4	0	5	5000000	5000000	Pass
10	4	1	5	500000	500000	Pass
11	4	2	5	50000	50000	Pass
12	4	3	5	5000	5000	Pass
13	4	5	5	5000	5000	Pass
14	4	6	1	2.20462	2.20462	Pass
15	4	7	1	35.274	35.274	Pass

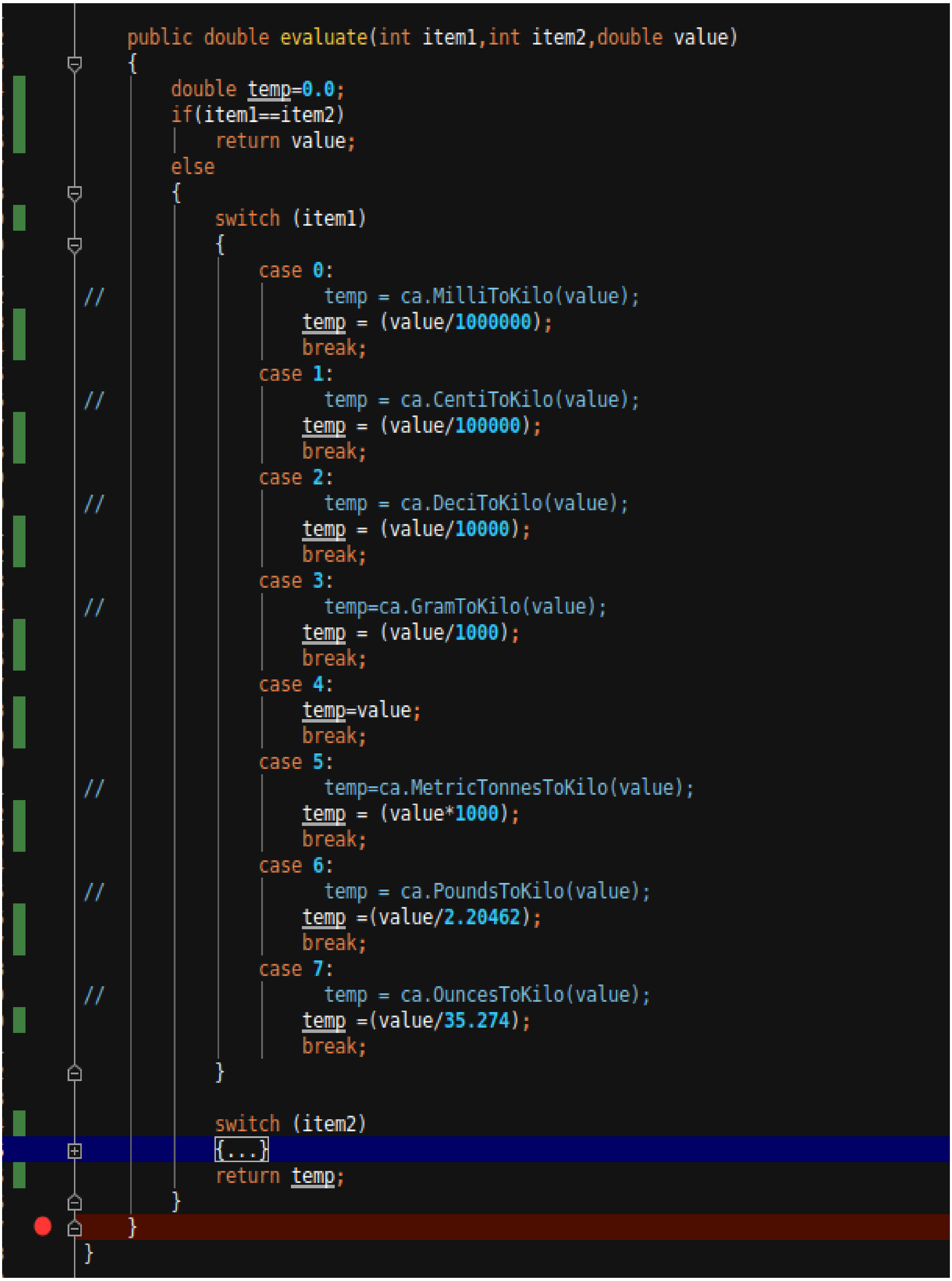


```
@Test
public void evaluate_KiloToKilo() throws Exception
{

    int choice1 = 4;
    int choice2 = 4;

    double val = 20.0;
    double expVal = 20.0;

    Assert.assertTrue("evaluate_KiloToKilo" + " failed !!",expVal == testObjectUnitWeight.evaluate(choice1, choice2, val));

}

@Test
public void evaluate_MilliToKilo() throws Exception
{

    int choice1 = 0;
    int choice2 = 4;

    double val = 2000000.0;
    double expVal = 2.0;

    Assert.assertTrue("evaluate_MilliToKilo" + " failed !!",expVal == testObjectUnitWeight.evaluate(choice1, choice2, val));

}
```



```
@Test
public void evaluate_KiloToMilli() throws Exception
{

    int choice1 = 4;
    int choice2 = 0;

    double val = 5.0;
    double expVal = 5000000.0;

    Assert.assertTrue("evaluate_KiloToMilli" + " failed !!",expVal == testObjectUnitWeight.evaluate(choice1, choice2, val));

}

@Test
public void evaluate_KiloToCenti() throws Exception
{

    int choice1 = 4;
    int choice2 = 1;

    double val = 5.0;
    double expVal = 500000.0;

    Assert.assertTrue("evaluate_KiloToCenti" + " failed !!",expVal == testObjectUnitWeight.evaluate(choice1, choice2, val));

}
```

Test Coverage



```
    public double evaluate(int item1,int item2,double value)
    {
        double temp=0.0;
        if(item1==item2)
            return value;
        else
        {
            switch (item1)
            {
                case 0:
//                    temp = ca.MilliToKilo(value);
                    temp = (value/1000000);
                    break;
                case 1:
//                    temp = ca.CentiToKilo(value);
                    temp = (value/100000);
                    break;
                case 2:
//                    temp = ca.DeciToKilo(value);
                    temp = (value/10000);
                    break;
                case 3:
//                    temp=ca.GramToKilo(value);
                    temp = (value/1000);
                    break;
                case 4:
                    temp=value;
                    break;
                case 5:
//                    temp=ca.MetricTonnesToKilo(value);
                    temp = (value*1000);
                    break;
                case 6:
//                    temp = ca.PoundsToKilo(value);
                    temp =(value/2.20462);
                    break;
                case 7:
//                    temp = ca.OuncesToKilo(value);
                    temp =(value/35.274);
                    break;
            }

            switch (item2)
            {...}
            return temp;
        }
    }
}
```

Contribution of Members

Rajeev Pankaj Shukla (MT2018091) – Application development and Testing for Combination view, Unit Weight Converter view and Unit Area Converter View and Integration.

Ravindra Singh Pawar (MT2018093) - Application development and Testing for Factorial view, Unit Length Converter view and Unit Temperature Converter View and Integration.