Software Testing
Project Report

Session: Spring 2021

Danish Hasan MSCS-20001

Abu Bakar MSCS-20013

Musa Khan MSCS-20065

Awais MSCS-20074

Employee Time Reporting

Department of Computer Science

Information Technology University Lahore

Pakistan

Project Contribution

Member	Code	Report
Danish	Wrote test cases in test suites test_1 – test_10 total 16 test cases written. Setup combined Junit eclipse workspace in our code repository.	Made report for test cases in test_1 – test_10
Abu Bakar	Wrote test cases in test suites testBinaryGCD_1 testBinaryGCD_2 testdivWord_1 testdivWord_2 testdivWord_3 total 17 test cases written Compiled all the code in a single file.	Made report for test cases in testBinaryGCD_1 testBinaryGCD_2 testdivWord_1 testdivWord_2 testdivWord_3
Awais	Wrote test cases in test suites MathContextTest_1 MathContextTest_2 MathContextTest_3 passesMillerRabinTest1 passesMillerRabinTest2 passesMillerRabinTest3 passesMillerRabinTest4 subtractTest1 subtractTest2 subtractTest3	Made report for test cases in MathContextTest_1 MathContextTest_2 MathContextTest_3 passesMillerRabinTest1 passesMillerRabinTest2 passesMillerRabinTest3 passesMillerRabinTest4 subtractTest1 subtractTest2 subtractTest3

	total 10 test cases written	
Musa	No contribution	No contribution

White-Box Testing with Junit

FUNCTION 1:

Encodes a byte array into Base64 format.

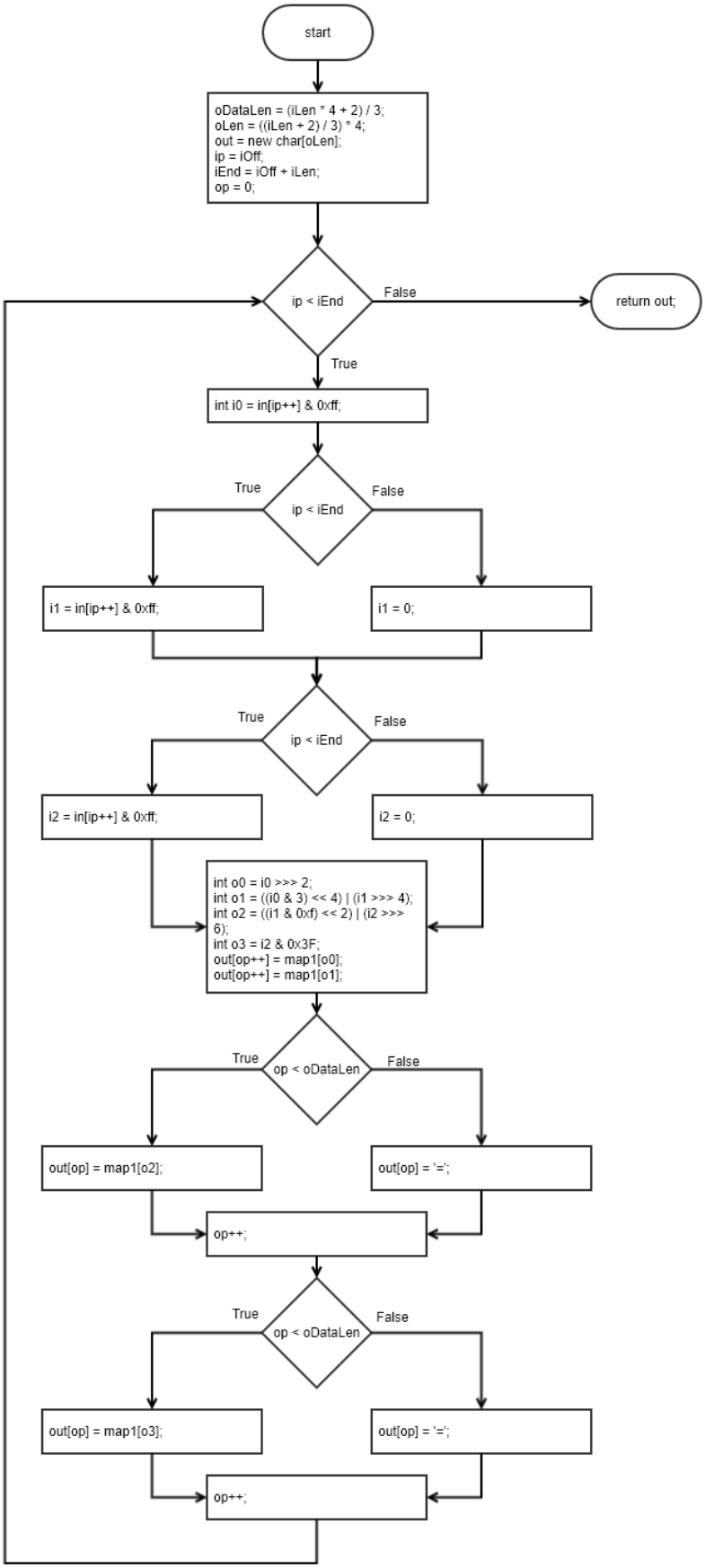
Note: map[] table is populated in another constructor function.

Source Code:

timesheet-master\src\main\java\timeSheet\util\properties\Base64Coder.java

```java
    public char[] encode(byte[] in, int iOff, int iLen) {
        int oDataLen = (iLen * 4 + 2) / 3;       // output length without padding
        int oLen = ((iLen + 2) / 3) * 4;         // output length including padding
        char[] out = new char[oLen];
        int ip = iOff;
        int iEnd = iOff + iLen;
        int op = 0;
        while (ip < iEnd) {
            int i0 = in[ip++] & 0xff;
            int i1 = ip < iEnd ? in[ip++] & 0xff : 0;
            int i2 = ip < iEnd ? in[ip++] & 0xff : 0;
            int o0 = i0 >>> 2;
            int o1 = ((i0 & 3) << 4) | (i1 >>> 4);
            int o2 = ((i1 & 0xf) << 2) | (i2 >>> 6);
            int o3 = i2 & 0x3F;
            out[op++] = map1[o0];
            out[op++] = map1[o1];
            out[op] = op < oDataLen ? map1[o2] : '=';
            op++;
            out[op] = op < oDataLen ? map1[o3] : '=';
            op++;
        }
        return out;
    }
```

CFG:


start
oDataLen = (iLen * 4 + 2) / 3;
oLen = ((iLen + 2) / 3) * 4;
out = new char[oLen];
ip = iOff;
iEnd = iOff + iLen;
op = 0;
ip < iEnd
False
return out;
True
int i0 = in[ip++] & 0xff;
ip < iEnd
True
False
i1 = in[ip++] & 0xff;
i1 = 0;
ip < iEnd
True
False
i2 = in[ip++] & 0xff;
i2 = 0;
int o0 = i0 >>> 2;
int o1 = ((i0 & 3) << 4) | (i1 >>> 4);
int o2 = ((i1 & 0xf) << 2) | (i2 >>>
6);
int o3 = i2 & 0x3F;
out[op++] = map1[o0];
out[op++] = map1[o1];
op < oDataLen
True
False
out[op] = map1[o2];
out[op] = '=';
op++;
op < oDataLen
True
False
out[op] = map1[o3];
out[op] = '=';
op++;

Statement Coverage:

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 3;
		QUJD	QUJD	Pass	Covers all statements

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 1:
     * timesheet-master/src/main/java/timeSheet/util/properties/Base64Coder.java
     * Test Type: Statement Coverage (covers all statements)
     *****************************************************************/
    @Test
    void test_5() {
        System.out.println("test_5");

        byte[] in = new byte[]{'A', 'B', 'C'};
        byte[] expected_out = new byte[]{'Q', 'U', 'J', 'D'};

        Base64Coder c = new Base64Coder();

        char[] ret = c.encode(in, 0, 3);

        assertEquals(ret.length, 4);

        for(int i = 0; i < ret.length; ++i)
        {
            assertEquals(ret[i], expected_out[i]);
        }
    }
```

Test Case Result:

Branch Coverage:

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 1;
		QQ==	QQ==	Pass	Covers 66TF, 68F, 69F, 76F, 78F

Test Case Code:

```java
/*****************************************************************
 * FUNCTION 1:
 * timesheet-master/src/main/java/timeSheet/util/properties/Base64Coder.java
 * Test Type: Branch Coverage (covers cases to generate padding i.e. = character in output)
 *****************************************************************/
@Test
void test_6() {
    System.out.println("test_6");

    byte[] in = new byte[]{'A', 'B', 'C'};
    byte[] expected_out = new byte[]{'Q', 'Q', '=', '='};

    Base64Coder c = new Base64Coder();

    char[] ret = c.encode(in, 0, 1);

    assertEquals(ret.length, 4);

    for(int i = 0; i < ret.length; ++i)
    {
        assertEquals(ret[i], expected_out[i]);
    }
}
```

Test Case Result:

test_6() (0.001 s)

Basis Path:

Edges - Nodes + 2 = 22 – 18 + 2 = 6

Path 1: 66F

Path 2: 66T, 68T, 69T, 76T, 78T

Path 3: 66T, 68T, 69F, 76T, 78F

Path 4: 66T, 68F, 69F, 76F, 78F

Path 5: 66T, 68F, 69F, 76F, 78T

Path 6: 66T, 68F, 69T, 76F, 78F

Note that no logical path is possible to cause 69T while 68F. Same is the case with 76F and 78T. Similarly, conditions in 76 and 78 also depend upon the same factor as 68, 69 so it is not possible for 68T but 76F and vice versa.

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks

1	In[] = {'A', 'B', 'C'};
	iOff = 0;
	iLen = 0;
		Empty String	Empty String	Pass	Covers Path1

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 1:
     * timesheet-master/src/main/java/timeSheet/util/properties/Base64Coder.java
     * Test Type: Basis Path (skips loop completely)
     *****************************************************************/
    @Test
    void test_7() {
        System.out.println("test_7");

        byte[] in = new byte[]{'A', 'B', 'C'};

        Base64Coder c = new Base64Coder();

        char[] ret = c.encode(in, 0, 0);

        assertEquals(ret.length, 0);
    }
```

Test Case Result:

test_7() (0.001 s)

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	iLen	59	60, 61, 64
2	oLen	61	62
3	Op	65, 74, 75, 77, 79	74, 75, 76, 77, 78, 79

Variable #	Variable Name	DU pairs
1	iLen	<59, 60>, <59, 61>, <59, 64>
2	oLen	<61, 62>
3	Op	<65,74>, <74,75>, <75,76>, <75,77>, <77,78>, <77,79>, <79,74>

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks

1	In[] = {'A', 'B', 'C', 'D', 'E', 'F'};
	iOff = 0;
	iLen = 6;
		QUJDREVG	QUJDREVG	Pass	iLen = Covers <59, 60>, <59, 61>, <59, 64>
					oLen = Covers <61, 62>
					op = Covers <65,74>, <74,75>, <75,76>, <75,77>, <77,78>, <77,79>, <79,74>

Test Case Code:

```java
/*****************************************************************
 * FUNCTION 1:
 * timesheet-master/src/main/java/timeSheet/util/properties/Base64Coder.java
 * Test Type: Data flow (covers define use pairs for variables iLen, oLen and Op)
 *****************************************************************/
@Test
void test_8() {
    System.out.println("test_8");

    byte[] in = new byte[]{'A', 'B', 'C', 'D', 'E', 'F'};
    byte[] expected_out = new byte[]{'Q', 'U', 'J', 'D', 'R', 'E', 'V', 'G'};

    Base64Coder c = new Base64Coder();

    char[] ret = c.encode(in, 0, 6);

    assertEquals(ret.length, 8);

    for(int i = 0; i < ret.length; ++i)
    {
        assertEquals(ret[i], expected_out[i]);
    }
}
```

Test Case Result:

FUNCTION 3:

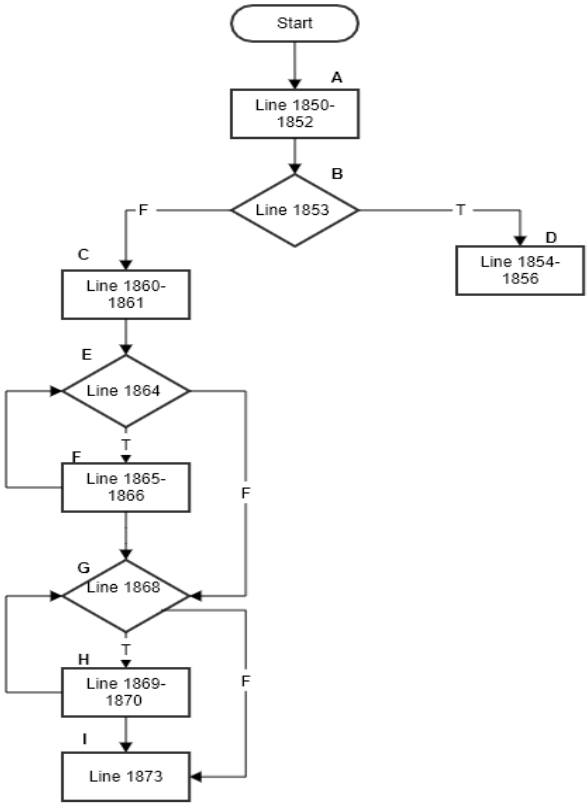
Source Code:

https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java

```java
    static final long LONG_MASK = 0xffffffffL;
    static long divWord(long n, int d) {
        long dLong = d & LONG_MASK;
        long r;
        long q;
        if (dLong == 1) {
            q = (int)n;
            r = 0;
            return (r << 32) | (q & LONG_MASK);
        }

        // Approximate the quotient and remainder
        q = (n >>> 1) / (dLong >>> 1);
        r = n - q*dLong;

        // Correct the approximation
        while (r < 0) {
            r += dLong;
            q--;
        }
        while (r >= dLong) {
            r -= dLong;
            q++;
        }
        // n - q*dlong == r && 0 <= r <dLong, hence we're done.
        return (r << 32) | (q & LONG_MASK);
    }

```

```java
    /*****************************************************************
     * FUNCTION 3:
     * https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java
     *****************************************************************/

    static final long LONG_MASK = 0xffffffffL;
    public long divWord(long n, int d) {
        long dLong = d & LONG_MASK;
        long r;
        long q;
        if (dLong == 1) {
            q = (int)n;
            r = 0;
            return (r << 32) | (q & LONG_MASK);
        }

        // Approximate the quotient and remainder
        q = (n >>> 1) / (dLong >>> 1);
        r = n - q*dLong;

        // Correct the approximation
        while (r < 0) {
            r += dLong;
            q--;
        }
        while (r >= dLong) {
            r -= dLong;
            q++;
        }
        // n - q*dlong == r && 0 <= r <dLong, hence we're done.
        return (r << 32) | (q & LONG_MASK);
    }

    /*****************************************************************
```

CFG:



Test Case Code:

```
/*****************************************************************
 * FUNCTION 3:
 * https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java
 * Test Type: Branch Coverage
 *****************************************************************/
@Test
// Covers B1853T
void testdivWord_1 (){
    MutableBigInteger M = new MutableBigInteger();
    long actual = M.divWord(16,1);
    long expected = 16;
    assertEquals(actual, expected);
}

/*****************************************************************
 * FUNCTION 3:
 * https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java
 * Test Type: Branch Coverage
 *****************************************************************/
@Test
// Covers B1853F , B1864TF, B1864F
void testdivWord_2 (){
    MutableBigInteger M = new MutableBigInteger();
    long actual = M.divWord(10,3);

    // workaround because this much long value is causing the error.
    Long lobject = new Long("4294967299");
    long expected = lobject.longValue();
    assertEquals(actual, expected);
}
```

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n = 16 d = 1	16	16	Pass	Covers Statement 1850-1857
2	n = 10 d = 3	4294967299	4294967299	Pass	Covers Statement 1850,1851,1852, 1860-1868, 1873
3	-	-	-	-	Statement 1869- 1870 I think this is a dead code, I could not find any such case in which the condition at 1868 becomes True

Test Result

testdivWord_1() (0.001 s)
testdivWord_2() (0.003 s)

Loop Boundary:

I think Loop at line 1868 is a dead code, I could not find any such case in which the condition at 1868 becomes True.

Test cases are only for the loop at line 1864.

I choose loop upper bound = 5

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	n =10 d = 5	2	2	Pass	Loop at line 1864 is skipped entirely.
2	n =5 d = 3	8589934593	8589934593	Pass	Loop at line 1864 is run only once
3	n =20 d =3	8589934598	8589934598	Pass	Loop at line 1864 is run 4 times

4	n = 28 d = 3	4294967305	4294967305	Pass	Loop at line 1864 is run 5 times.
5	n = 32 d = 3	8589934602	8589934602	Pass	Loop at line 1864 is run 6 times.

Test Case Code:

```java
801     /*****************************************************************
802      * FUNCTION 3:
803      * https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java
804      * Test Type: Loop Boundary
805      *****************************************************************/
806     @ParameterizedTest
807     @CsvSource(value = { "10,5,2","5,3,8589934593", "20,3,8589934598","28,3,4294967305","32,3,8589934602"})
808     // Test cases are only for the loop at line 1864.
809     // Loop is skipped entirely, Loop is run once,
810     // Choose loop upper bound N=5
811     // Loop is run 4 times, Loop is run 5 times, Loop is run 6 times
812     void testdivWord_3 (long n, int d, long expected){
813         MutableBigInteger M = new MutableBigInteger();
814         long actual = M.divWord(n,d);
815
816         assertEquals(actual, expected);
817     }
818
```

Test Result:

- testdivWord_3(long, int, long) (0.001 s)
  - [1] 10, 5, 2 (0.001 s)
  - [2] 5, 3, 8589934593 (0.001 s)
  - [3] 20, 3, 8589934598 (0.002 s)
  - [4] 28, 3, 4294967305 (0.002 s)
  - [5] 32, 3, 8589934602 (0.000 s)

FUNCTION 4:

Decodes a byte array from Base64 format.

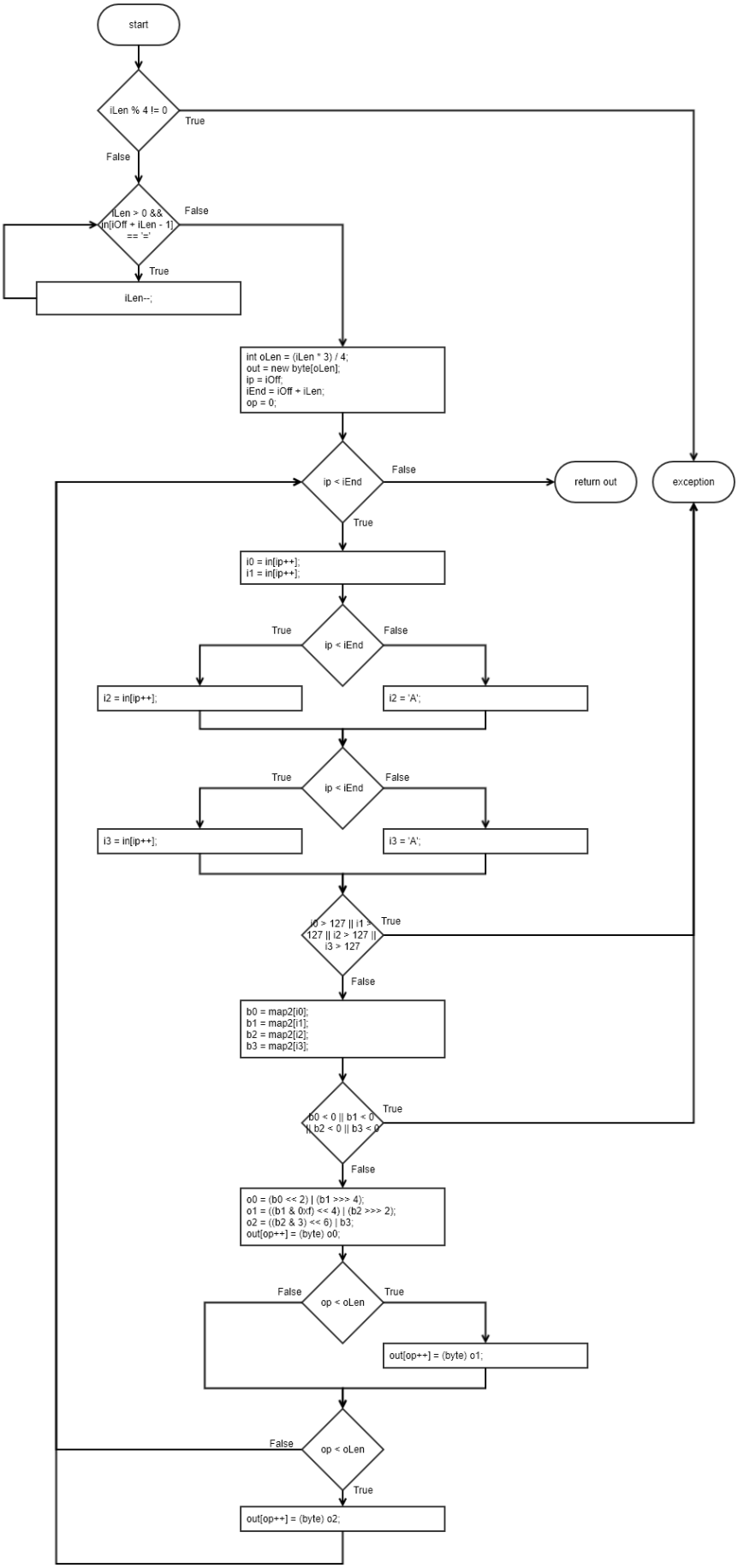
Note: map2[] table is populated in another constructor function.

Source Code:

timesheet-master\src\main\java\timeSheet\util\properties\Base64Coder.java

```java
    public byte[] decode(char[] in, int iOff, int iLen) {
        if (iLen % 4 != 0)
            throw new IllegalArgumentException("Length of Base64 encoded input string is not a multiple of 4.");
        while (iLen > 0 && in[iOff + iLen - 1] == '=') iLen--;
        int oLen = (iLen * 3) / 4;
        byte[] out = new byte[oLen];
        int ip = iOff;
        int iEnd = iOff + iLen;
        int op = 0;
        while (ip < iEnd) {
            int i0 = in[ip++];
            int i1 = in[ip++];
            int i2 = ip < iEnd ? in[ip++] : 'A';
            int i3 = ip < iEnd ? in[ip++] : 'A';
            if (i0 > 127 || i1 > 127 || i2 > 127 || i3 > 127)
                throw new IllegalArgumentException("Illegal character in Base64 encoded data.");
            int b0 = map2[i0];
            int b1 = map2[i1];
            int b2 = map2[i2];
            int b3 = map2[i3];
            if (b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0)
                throw new IllegalArgumentException("Illegal character in Base64 encoded data.");
            int o0 = (b0 << 2) | (b1 >>> 4);
            int o1 = ((b1 & 0xf) << 4) | (b2 >>> 2);
            int o2 = ((b2 & 3) << 6) | b3;
            out[op++] = (byte) o0;
            if (op < oLen) out[op++] = (byte) o1;
            if (op < oLen) out[op++] = (byte) o2;
        }
        return out;
    }
}
```

CFG:


start
iLen % 4 != 0
True
False
iLen > 0 &&
in[iOff + iLen - 1]
== '='
False
True
iLen--;
int oLen = (iLen * 3) / 4;
out = new byte[oLen];
ip = iOff;
iEnd = iOff + iLen;
op = 0;
ip < iEnd
False
return out
exception
True
i0 = in[ip++];
i1 = in[ip++];
ip < iEnd
True
False
i2 = in[ip++];
i2 = 'A';
ip < iEnd
True
False
i3 = in[ip++];
i3 = 'A';
i0 > 127 || i1 >
127 || i2 > 127 ||
i3 > 127
True
False
b0 = map2[i0];
b1 = map2[i1];
b2 = map2[i2];
b3 = map2[i3];
b0 < 0 || b1 < 0
|| b2 < 0 || b3 < 0
True
False
o0 = (b0 << 2) | (b1 >>> 4);
o1 = ((b1 & 0xf) << 4) | (b2 >>> 2);
o2 = ((b2 & 3) << 6) | b3;
out[op++] = (byte) o0;
op < oLen
False
True
out[op++] = (byte) o1;
op < oLen
False
True
out[op++] = (byte) o2;

Boundary Interior:

Exception cases are not covered under sir's guidance.
Possible logical paths:

- A: 118T->119T-> 132T-> 133T
- B: 118T-> 119F-> 132T->133F
- C: 118F-> 119F-> 132T-> 133F

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	Covers Path A
2	In[] = 'QQ==' iOff = 0 iLen = 4	'A'	'A'	Pass	Covers Path B
3	In[] = 'QUI=' iOff = 0 iLen = 4	'AB'	'AB'	Pass	Covers Path C

Test Case Code:

```java
/*****************************************************************
 * FUNCTION 4:
 * timesheet-master/src/main/java/timeSheet/util/properties/Base64Coder.java
 * Test Type: Boundary Interior (covers all possible paths within loop)
 *****************************************************************/
@Test
void test_9() {
    System.out.println("test_9");

    {
        System.out.println("covers path A");

        char[] in = new char[]{'Q', 'U', 'J', 'D'};
        byte[] expected_out = new byte[]{'A', 'B', 'C'};

        Base64Coder c = new Base64Coder();

        byte[] ret = c.decode(in, 0, 4);

        assertEquals(ret.length, 3);

        for(int i = 0; i < ret.length; ++i)
        {
            assertEquals(ret[i], expected_out[i]);
        }
    }

    {
        System.out.println("covers path B");

        char[] in = new char[]{'Q', 'U', '=', '='};
        byte[] expected_out = new byte[]{'A'};

        Base64Coder c = new Base64Coder();

        byte[] ret = c.decode(in, 0, 4);

        assertEquals(ret.length, 1);

        for(int i = 0; i < ret.length; ++i)
        {
            assertEquals(ret[i], expected_out[i]);
        }
    }

    {
        System.out.println("covers path C");

        char[] in = new char[]{'Q', 'U', 'I', '='};
        byte[] expected_out = new byte[]{'A', 'B'};

        Base64Coder c = new Base64Coder();

        byte[] ret = c.decode(in, 0, 4);

        assertEquals(ret.length, 2);

        for(int i = 0; i < ret.length; ++i)
        {
            assertEquals(ret[i], expected_out[i]);
        }
    }
}
```

Test Case Result:

Loop Boundary:

Consider N=12 for loop. (Note that for valid input N-1 must be 8 and N+1 must be 16)

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks
1	In[] = 'QUJD' iOff = 0 iLen = 0	Empty String	Empty String	Pass	Covers 115F

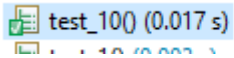
2	In[] = 'QUJD' iOff = 0 iLen = 4	'ABC'	'ABC'	Pass	Covers 115F once
3	In[] = 'QUJDREU=' iOff = 0 iLen = 8	'ABCDE'	'ABCDE'	Pass	Covers 115T for N-1
4	In[] = 'QUJDREVGRw===' iOff = 0 iLen = 12	'ABCDEFG'	'ABCDEFG'	Pass	Covers 115T for N
5	In[] = 'QUJDREVGR0hJSg==' iOff = 0 iLen = 16	'ABCDEFGHIJ'	'ABCDEFGHIJ'	Pass	Covers 115T for N+1

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 4:
     * timesheet-master/src/main/java/timeSheet/util/properties/Base64Coder.java
     * Test Type: Loop Boundary (Consider N=12 for loop. (Note that for valid input N-1 must be 8 and N+1 must be 16)
     *****************************************************************/
    @Test
    void test_10() {
        System.out.println("test_10");

        {
            System.out.println("covers loop 0 times");

            char[] in = new char[]{'Q', 'U', 'J', 'D'};

            Base64Coder c = new Base64Coder();

            byte[] ret = c.decode(in, 0, 0);

            assertEquals(ret.length, 0);
        }

        {
            System.out.println("covers loop once");

            char[] in = new char[]{'Q', 'U', 'J', 'D'};
            byte[] expected_out = new byte[]{'A', 'B', 'C'};

            Base64Coder c = new Base64Coder();

            byte[] ret = c.decode(in, 0, 4);

            assertEquals(ret.length, 3);

            for(int i = 0; i < ret.length; ++i)
            {
                assertEquals(ret[i], expected_out[i]);
            }
        }

        {
            System.out.println("covers loop for N-1");

            char[] in = new char[]{'Q', 'U', 'J', 'D', 'R', 'E', 'U', '='};
            byte[] expected_out = new byte[]{'A', 'B', 'C', 'D', 'E'};

            Base64Coder c = new Base64Coder();

            byte[] ret = c.decode(in, 0, 8);

            assertEquals(ret.length, 5);

            for(int i = 0; i < ret.length; ++i)
            {
                assertEquals(ret[i], expected_out[i]);
            }
        }
```

```java
        {
            System.out.println("covers loop for N");

            char[] in = new char[]{'Q', 'U', 'J', 'D', 'R', 'E', 'V', 'G', 'R', 'w', '=', '='};
            byte[] expected_out = new byte[]{'A', 'B', 'C', 'D', 'E', 'F', 'G'};

            Base64Coder c = new Base64Coder();

            byte[] ret = c.decode(in, 0, 12);

            assertEquals(ret.length, 7);

            for(int i = 0; i < ret.length; ++i)
            {
                assertEquals(ret[i], expected_out[i]);
            }
        }

        {
            System.out.println("covers loop for N+1");

            char[] in = new char[]{'Q', 'U', 'J', 'D', 'R', 'E', 'V', 'G', 'R', '0', 'h', 'J', 'S', 'g', '=', '='};
            byte[] expected_out = new byte[]{'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};

            Base64Coder c = new Base64Coder();

            byte[] ret = c.decode(in, 0, 16);

            assertEquals(ret.length, 10);

            for(int i = 0; i < ret.length; ++i)
            {
                assertEquals(ret[i], expected_out[i]);
            }
        }
    }
```

Test Case Result:



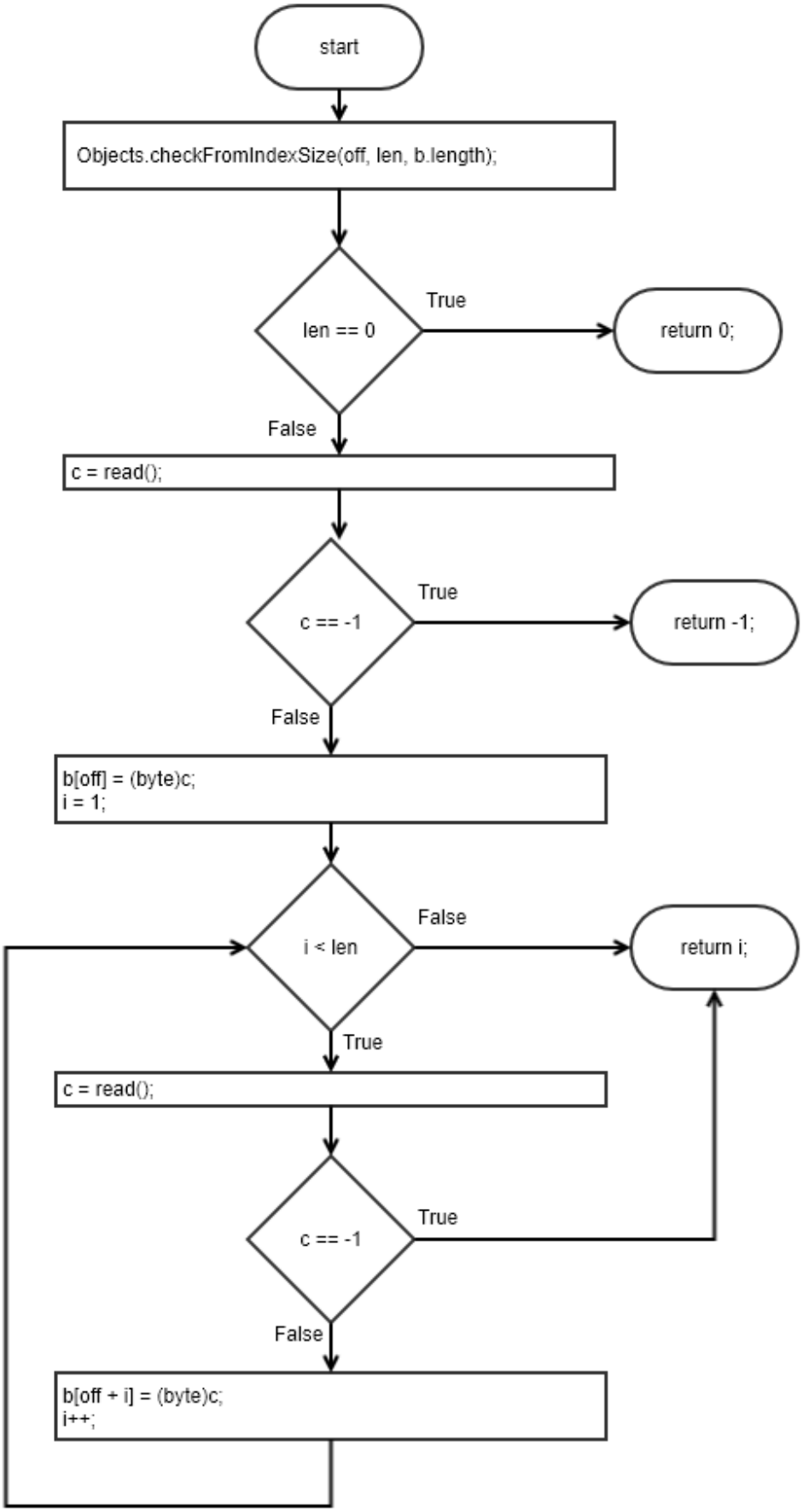
FUNCTION 5:

Source Code:

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/io/InputStream.
Java
checkFromIndexSize and read are external APIs. checkFromIndexSize can be implemented as dummy stub while read is implemented as needed by each test case.

```java
    public int read(byte b[], int off, int len) throws IOException {
        Objects.checkFromIndexSize(off, len, b.length);
        if (len == 0) {
            return 0;
        }

        int c = read();
        if (c == -1) {
            return -1;
        }
        b[off] = (byte)c;

        int i = 1;
        try {
            for (; i < len ; i++) {
                c = read();
                if (c == -1) {
                    break;
                }
                b[off + i] = (byte)c;
            }
        } catch (IOException ee) {
        }
        return i;
    }
```

CFG:


start
Objects.checkFromIndexSize(off, len, b.length);
len == 0
True
return 0;
False
c = read();
c == -1
True
return -1;
False
b[off] = (byte)c;
i = 1;
i < len
False
return i;
True
c = read();
c == -1
True
False
b[off + i] = (byte)c;
i++;

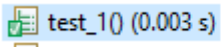
Statement Coverage:

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks
1	b[] = Empty Array off = 0 len = 3	3, b[] ='ABC'	3, b[] ='ABC'	Pass	External module API read() returns 'A', 'B', 'C' in consecutive calls.

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 5:
     * https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/io/InputStream.Java
     * Test Type: Statement Coverage
     *****************************************************************/
    @Test
    void test_1() {
        System.out.println("test_1");

        byte[] actual_A = new byte[10];
        byte[] expected_A = new byte[10];
        expected_A[0] = 'A';
        expected_A[1] = 'B';
        expected_A[2] = 'C';

        ReadClass c = new ReadClass();

        c.setExpectedRead(expected_A, 3);

        try {
            int ret = c.read(actual_A, 0, 3);

            assertEquals(ret, 3);

            for(int i = 0; i < actual_A.length; ++i)
            {
                assertTrue(actual_A[i] == expected_A[i]);
            }

        } catch (IOException e) {
            fail("test threw an exception");
        }
    }
```

Test Case Result:



Branch Coverage:

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks

1	b[] = Empty Array off = 0 len = 0	0, b[] = Empty Array	0, b[] = Empty Array	Pass	External module API read() is never called. 280T

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 5:
     * https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/io/InputStream.Java
     * Test Type: Branch Coverage
     *****************************************************************/
    @Test
    void test_2() {
        System.out.println("test_2");

        byte[] actual_A = new byte[10];
        byte[] expected_A = new byte[10];

        ReadClass c = new ReadClass();

        try {
            int ret = c.read(actual_A, 0, 0);

            assertEquals(ret, 0);

            for(int i = 0; i < actual_A.length; ++i)
            {
                assertTrue(actual_A[i] == expected_A[i]);
            }

        } catch (IOException e) {
            fail("test threw an exception");
        }
    }
```

Test Case Result:

test_2() (0.001 s)

Loop Boundary:

Consider N=4 for loop boundary

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks
1	b[] = Empty Array off = 0 len = 2	2, b[] ='AB'	2, b[] ='AB'	Pass	External module API read() returns 'A', 'B' in consecutive calls. Covers 292T once

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 5:
     * https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/io/InputStream.Java
     * Test Type: Loop Boundary (Covers loop once)
     *****************************************************************/
    @Test
    void test_3() {
        System.out.println("test_3");

        byte[] actual_A = new byte[10];
        byte[] expected_A = new byte[10];
        expected_A[0] = 'A';
        expected_A[1] = 'B';

        ReadClass c = new ReadClass();

        c.setExpectedRead(expected_A, 2);

        try {
            int ret = c.read(actual_A, 0, 2);

            assertEquals(ret, 2);

            for(int i = 0; i < actual_A.length; ++i)
            {
                assertTrue(actual_A[i] == expected_A[i]);
            }

        } catch (IOException e) {
            fail("test threw an exception");
        }
    }
```

Test Case Result:

test_3() (0.001 s)

Basis Path:

Decision points + 1 = 4 + 1 = 5

Path 1: 280T

Path 2: 280F, 285T

Path 3: 280F, 285F, 292F

Path 4: 280F, 285F, 292TF, 294F

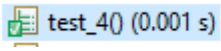
Path 5: 280F, 285F, 292T, 294T

Test case#	Input	Expected Output	Actual Output	Result	Comments / Remarks
1	b[] = Empty Array off = 0 len = 3	-1, b[] = Empty Array	-1, b[] = Empty Array	Pass	External module API read() returns -1 to notify an error at first call. **Covers Path2**

Test Case Code:

```java
    /*****************************************************************
     * FUNCTION 5:
     * https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/io/InputStream.Java
     * Test Type: Basis Path (stub read shall return -1 to trigger error in our read call)
     *****************************************************************/
    @Test
    void test_4() {
        System.out.println("test_4");

        byte[] actual_A = new byte[10];
        byte[] expected_A = new byte[10];

        ReadClass c = new ReadClass();

        try {
            int ret = c.read(actual_A, 0, 3);

            assertEquals(ret, -1);

            for(int i = 0; i < actual_A.length; ++i)
            {
                assertTrue(actual_A[i] == expected_A[i]);
            }

        } catch (IOException e) {
            fail("test threw an exception");
        }
    }
```

Test Case Result:



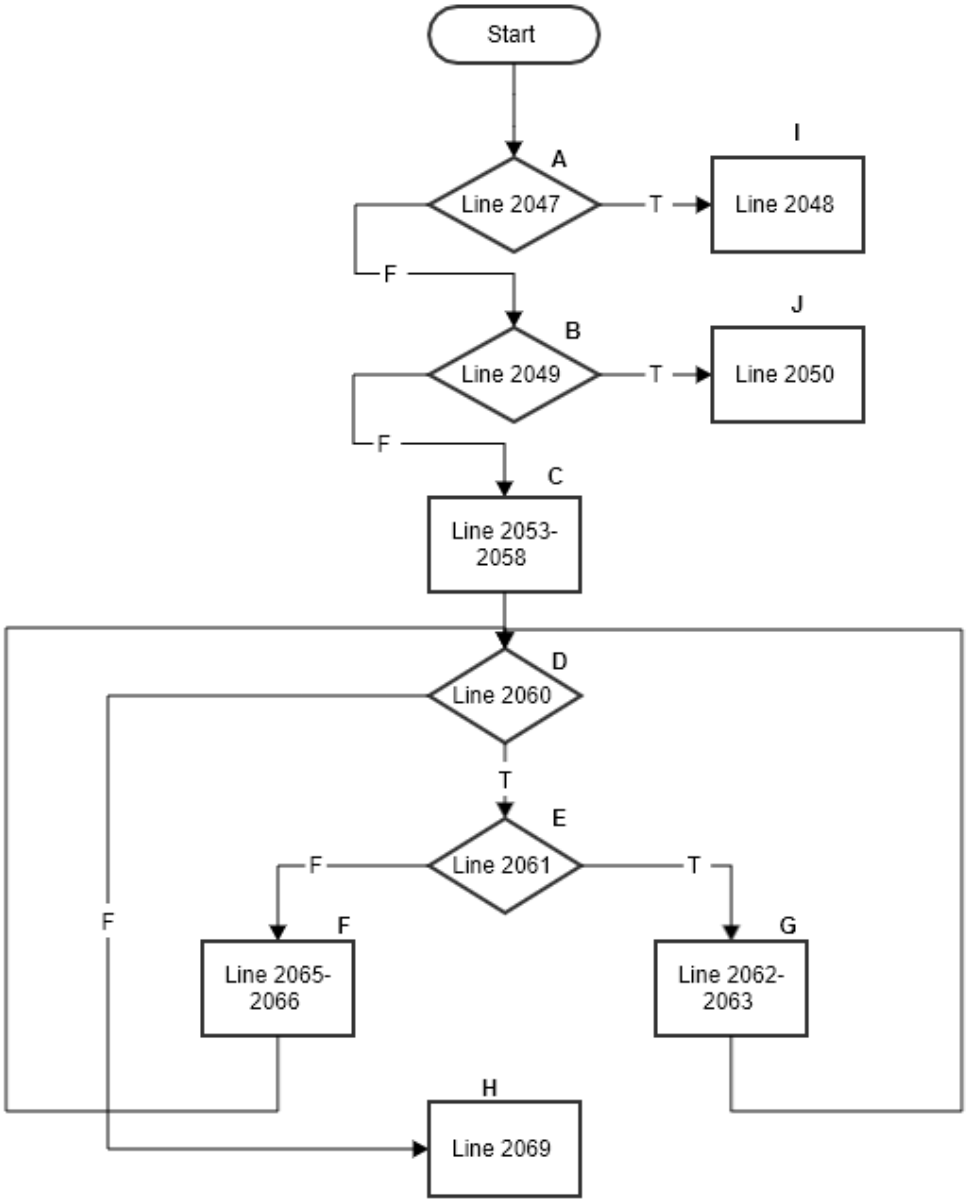
FUNCTION 6:

Source Code:

https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java

```java
    static int binaryGcd(int a, int b) {
        if (b == 0)
            return a;
        if (a == 0)
            return b;

        // Right shift a & b till their last bits equal to 1.
        int aZeros = Integer.numberOfTrailingZeros(a);
        int bZeros = Integer.numberOfTrailingZeros(b);
        a >>>= aZeros;
        b >>>= bZeros;

        int t = (aZeros < bZeros ? aZeros : bZeros);

        while (a != b) {
            if ((a+0x80000000) > (b+0x80000000)) {  // a > b as unsigned
                a -= b;
                a >>>= Integer.numberOfTrailingZeros(a);
            } else {
                b -= a;
                b >>>= Integer.numberOfTrailingZeros(b);
            }
        }
        return a<<t;
    }
```

CFG:



.

Statement Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers statement 2047-2048
2	a = 0 b =15	15	15	Pass	Covers statement 2049-2050
3	a = 98 b =56	14	14	Pass	Covers statement 2047,2049, 2051-2069

Branch Coverage:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers B2047T
2	a = 0 b =15	15	15	Pass	Covers B2049T, B2047F
3	a = 98 b =56	14	14	Pass	Covers B2047F, B2049F, B2060TF, B2061T
4	a = 56 b =98	14	14	Pass	Covers B2047F, B2049F, B2060TF, B2061F

Boundary Interior:

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 98 b =56	14	14	Pass	Covers boundary interior path DEG
2	a = 56 b =98	14	14	Pass	Covers boundary interior path DEF

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses

1	A	2046, 2055, 2062, 2063	2048, 2049, 2053, 2055, 2060, 2061, 2062, 2063, 2065, 2069
2	b	2046, 2056, 2065, 2066	2047, 2050, 2054, 2056, 2060, 2061, 2062, 2065, 2066
3	aZeros	2053	2055, 2058

Variable #	Variable Name	DU pairs
1	a	<2046, 2048> <2046, 2049> <2046, 2053> <2046, 2055> <2055, 2060> <2055, 2061> <2055, 2062> <2055, 2065> <2055, 2069> <2062, 2063> <2063, 2060> <2063, 2061> <2063, 2062> <2063, 2069>
2	b	<2046, 2047> <2046, 2050> <2046, 2054> <2046, 2056> <2056, 2060> <2056, 2061> <2056, 2062> <2056, 2065> <2065, 2066> <2066, 2060> <2066, 2061> <2066, 2062>
3	aZeros	<2053, 2055> <2053, 2058>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	For a covers <2046, 2048> For b covers <2046, 2047>
2	a = 0 b =15	15	15	Pass	For a covers <2046, 2049> For b covers <2046, 2047> <2046, 2050>
3	a = 98 b =56	14	14	Pass	For a covers <2046, 2049> <2046, 2053> <2046, 2055> <2055, 2060> <2055, 2061> <2055, 2062> <2062, 2063>

					<2063, 2060> <2063, 2061> <2063, 2062> <2063, 2069> For b covers <2046, 2047> <2046, 2054> <2046, 2056> <2056, 2060> <2056, 2061> <2056, 2062> For aZeros covers: <2053, 2055> <2053, 2058>
4	a = 56 b =98	14	14	Pass	For a covers <2046, 2049> <2046, 2053> <2046, 2055> <2055, 2060> <2055, 2061> <2055, 2065> <2055, 2069> For b covers <2046, 2047> <2046, 2054> <2046, 2056> <2056, 2060> <2056, 2061> <2056, 2065> <2065, 2066> <2066, 2060> <2066, 2061> <2066, 2062> For aZeros covers: <2053, 2055> <2053, 2058>

Test Code

```java
    /*****************************************************************
     * FUNCTION 6:
     * https://github.com/openjdk/jdk/tree/master/src/java.base/share/classes/java/math/MutableBigInteger.java
     * Test Type: Branch Coverage, Statement Coverage, Boundary Interior, , Data-Flow Testing (a,b,aZeros)
     *****************************************************************/
    @ParameterizedTest
    @CsvSource(value = { "15,0,15","0,15,15","98,56,14","56,98,14" })
    // Covers B2047T,
    // Covers B2049T, B2047F
    // Covers B2047F, B2049F, B2060TF, B2061T
    // Covers B2047F, B2049F, B2060TF, B2061F
    void testbinaryGCD_1 (int a, int b, int expected){
        MutableBigInteger M = new MutableBigInteger();

        int actual = M.binaryGcd(a,b);
        assertEquals(actual, expected);
    }
```

Test Result

- testbinaryGCD_1(int, int, int) (0.017 s)
  - [1] 15, 0, 15 (0.017 s)
  - [2] 0, 15, 15 (0.002 s)
  - [3] 98, 56, 14 (0.002 s)
  - [4] 56, 98, 14 (0.003 s)

Loop Boundary:

I choose loop upper bound = 5

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 12 b = 12	12	12	Pass	Loop is skipped entirely.
2	a = 4 b = 2	2	2	Pass	Loop is run only once
3	a = 6 b = 2	2	2	Pass	Loop is run twice.
4	a = 10 b = 2	2	2	Pass	Loop is run 4 times
5	a = 12 b = 2	2	2	Pass	Loop is run 5 times.
6	a = 14 b = 2	2	2	Pass	Loop is run 6 times.

Basis Path:

No. of Basis Paths = No. of decision points + 1

No. of Basis Paths = 4 + 1 = 5

Path 1: AI

Path 2: ABJ

Path 3: ABCDH

Path 4: ABCDEFH

Path 5: ABCDEGH

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	a = 15 b = 0	15	15	Pass	Covers basis path AI
2	a = 0 b =15	15	15	Pass	Covers basis path ABJ
3	a = 12 b = 12	12	12	Pass	Covers basis path ABCDH
4	a = 2 b = 4	2	2	Pass	Covers basis path ABCDEFH
5	a = 4 b = 2	2	2	Pass	Covers basis path ABCDEFH

Test Result

- testbinaryGCD_2(int, int, int) (0.002 s)
  - [1] 12, 12, 12 (0.002 s)
  - [2] 4, 2, 2 (0.002 s)
  - [3] 6, 2, 2 (0.002 s)
  - [4] 10, 2, 2 (0.001 s)
  - [5] 12, 2, 2 (0.002 s)
  - [6] 14, 2, 2 (0.001 s)

FUNCTION 8:

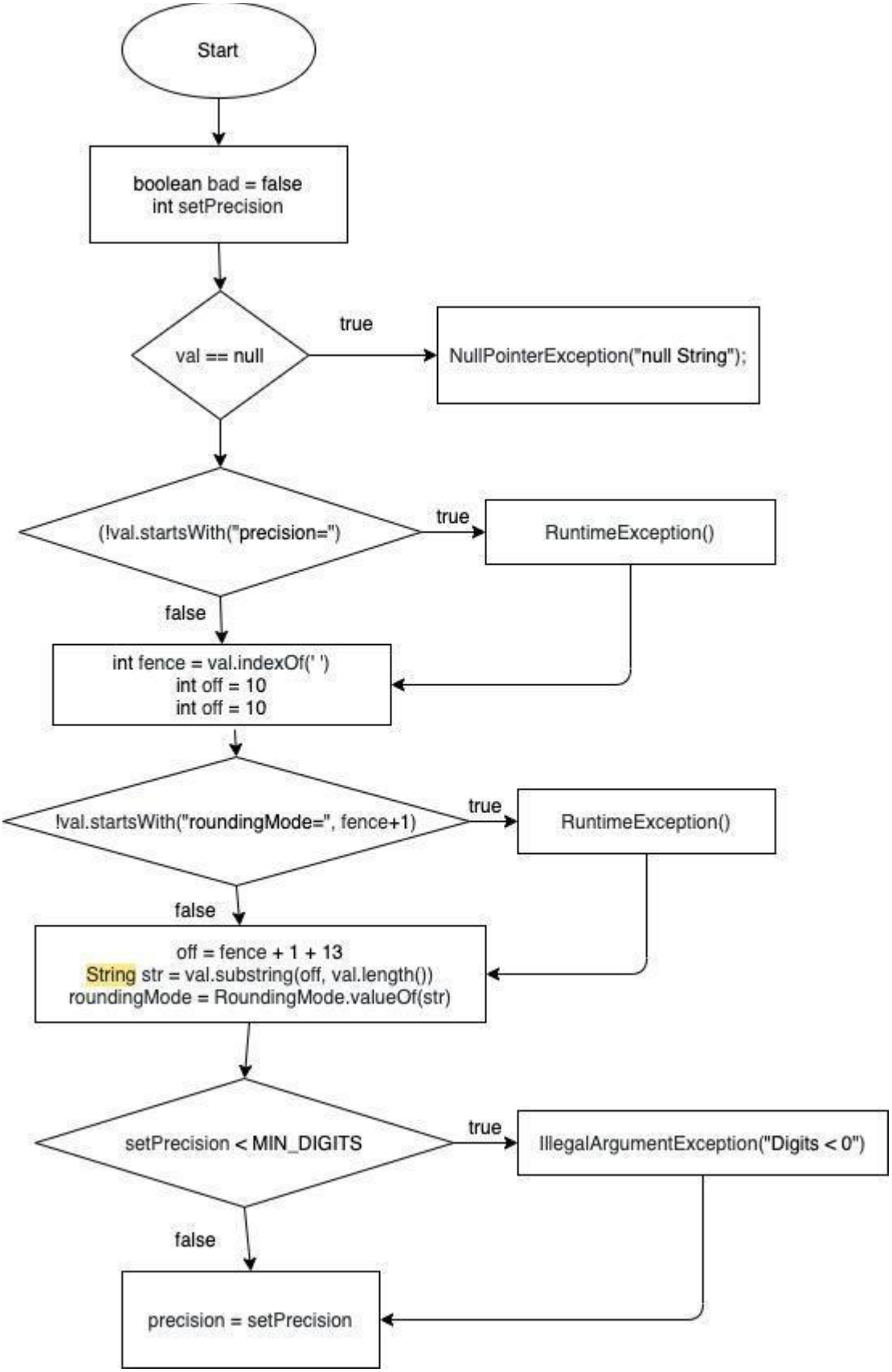
Source Code:

```java
    public MathContext(String val) {
        boolean bad = false;
        int setPrecision;
        if (val == null)
            throw new NullPointerException("null String");
        try { // any error here is a string format problem
            if (!val.startsWith("precision=")) throw new RuntimeException();
            int fence = val.indexOf(' ');    // could be -1
            int off = 10;                     // where value starts
            setPrecision = Integer.parseInt(val.substring(10, fence));

            if (!val.startsWith("roundingMode=", fence+1))
                throw new RuntimeException();
            off = fence + 1 + 13;
            String str = val.substring(off, val.length());
            roundingMode = RoundingMode.valueOf(str);
        } catch (RuntimeException re) {
            throw new IllegalArgumentException("bad string format");
        }

        if (setPrecision < MIN_DIGITS)
            throw new IllegalArgumentException("Digits < 0");
        // the other parameters cannot be invalid if we got here
        precision = setPrecision;
    }
```

Test Code:

```java
@Test
void MathContextTest1() {
    assertThrows(NullPointerException.class, () -> mu.MathContext(null), "No value is being passed inside method");
}

@Test
void MathContextTest2() {
    assertThrows(RuntimeException.class, () -> mu.MathContext("Second test function"), "Exception Raises");
}

@Test
void MathContextTest3() {
    assertThrows(IllegalArgumentException.class, () -> mu.MathContext("Precision=Second test function"), "Exception Raises");
}
```

CFG:


Start
boolean bad = false
int setPrecision
val == null
true
NullPointerException("null String");
(!val.startsWith("precision=")
true
RuntimeException()
false
int fence = val.indexOf(' ')
int off = 10
int off = 10
!val.startsWith("roundingMode=", fence+1)
true
RuntimeException()
false
off = fence + 1 + 13
String str = val.substring(off, val.length())
roundingMode = RoundingMode.valueOf(str)
setPrecision < MIN_DIGITS
true
IllegalArgumentException("Digits < 0")
false
precision = setPrecision

Statement Coverage:

Test case#	Input	Expected Output	Comments/Remarks
1	null	exception	Covered 184, 185, 186, 187
2	Second test function	exception	Covered 184, 185, 186, 188, 189
3	'precision=Second test function'	exception	Covered 184, 185, 186, 188, 190, 191, 192, 194, 195

Branch Coverage:

Test case#	Input	Expected Output	Comments/Remarks
1	(null)	exception	Covered B186(True)
2	Second test function	exception	Covered B186(False), B189(True)
3	'precision=Second test function'	exception	Covered B186(False), B189(False), B194(True)

Test Result:

MathContextTest1() (0.001 s)
MathContextTest2() (0.001 s)
MathContextTest3() (0.002 s)

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Expected Output	Comments/Remarks

1	(null)	exception	Covered C186(True)

2	Second test function	exception	Covered C186(False), C189(True)
3	'precision=Second test function'	exception	Covered C186(False), C189(False), C194(True)

Boundary Interior:

No Loop in the program.

Loop Boundary:

No Loop in the program.

Basis Path:

No of decision points = 4

No. of basis path = No of decision points +1 = 4+1 = 5

Path 1:

183, 184, 185, 186, 203, 206

Path 2:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 196, 197, 198, 203, 206

Path 3:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 203, 206

Path 4:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 203, 204, 206

Path 5:

183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 203, 204, 206

Test case#	Input	Expected Output	Comments/Remarks
1	null	Exception	Covers Path 1
2	'precision=Second test function'	Exception	Covers Path 3
3	Second test function	Exception	Covers Path 5

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	Val	183	186,189,190,192,197
2	setPrecision	185,192	203,206
3	Fence	190	192,194

Variable #	Variable Name	DU pairs
1	Val	<183,186>,<183,189>,<183,190>,<183,192>,<183,197>
2	setPrecision	<192,203>,<192,206>
3	Fence	<190,192>,<190,194>

Test case#	Input	Expected Output	Comments/Remarks
1	Second test function	Exception	For Val: <183,186>,<183,189> For setPrecision: Not used For Fence: Not used because it does not contains 'precision=' at start
2	'precision=Second test function'	Exception	For Val: <183,186>,<183,189>,<183,190>,<183,192>

			For setPrecision: Not used For Fence: <190,192>,<190,194> It returns exception because when next if executes it'll not find 'roundingMode=' at start

FUNCTION 9

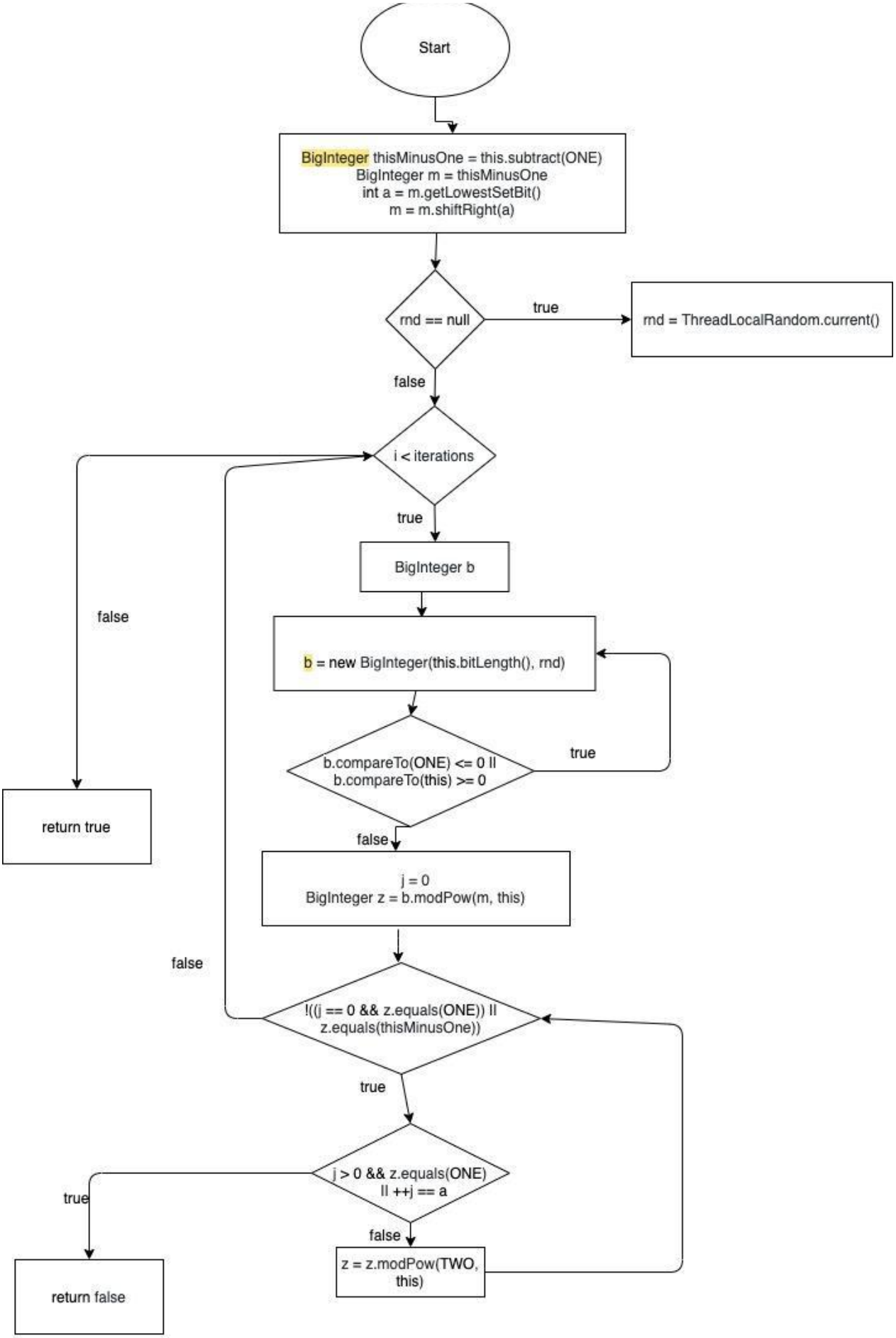
Source Code:

```
    private boolean passesMillerRabin(int iterations, Random rnd) {
        // Find a and m such that m is odd and this == 1 + 2**a * m
        BigInteger thisMinusOne = this.subtract(ONE);
        BigInteger m = thisMinusOne;
        int a = m.getLowestSetBit();
        m = m.shiftRight(a);

        // Do the tests
        if (rnd == null) {
            rnd = ThreadLocalRandom.current();
        }
        for (int i=0; i < iterations; i++) {
            // Generate a uniform random on (1, this)
            BigInteger b;
            do {
                b = new BigInteger(this.bitLength(), rnd);
            } while (b.compareTo(ONE) <= 0 || b.compareTo(this) >= 0);

            int j = 0;
            BigInteger z = b.modPow(m, this);
            while (!((j == 0 && z.equals(ONE)) || z.equals(thisMinusOne))) {
                if (j > 0 && z.equals(ONE) || ++j == a)
                    return false;
                z = z.modPow(TWO, this);
            }
        }
        return true;
    }
```

Test Case Code:

```java
@Test
void passesMillerRabinTest1() {
    Random rnd = new Random(12);
    assertTimeoutPreemptively(Duration.ofMillis(100), () -> {
        assertTrue(mu.passesMillerRabin(0, rnd), "Should be true");
    });
}

@Test
void passesMillerRabinTest2() {
    Random rnd = new Random();
    assertTimeoutPreemptively(Duration.ofMillis(100), () -> {
        assertTrue(mu.passesMillerRabin(-1, rnd), "Should be true");
    });
}

@Test
void passesMillerRabinTest3() {
    Random rnd = new Random(4);
    assertTrue(mu.passesMillerRabin(0, rnd), "Should be true");
}

@Test
void passesMillerRabinTest4() {
    Random rnd = new Random();
    assertTrue(mu.passesMillerRabin(0, null), "Should be true");
}
```

CFG:


Start
BigInteger thisMinusOne = this.subtract(ONE)
BigInteger m = thisMinusOne
int a = m.getLowestSetBit()
m = m.shiftRight(a)
rnd == null
true
rnd = ThreadLocalRandom.current()
false
i < iterations
true
BigInteger b
false
b = new BigInteger(this.bitLength(), rnd)
b.compareTo(ONE) <= 0 ||
b.compareTo(this) >= 0
true
return true
false
j = 0
BigInteger z = b.modPow(m, this)
false
!((j == 0 && z.equals(ONE)) ||
z.equals(thisMinusOne))
true
j > 0 && z.equals(ONE)
|| ++j == a
true
false
z = z.modPow(TWO,
this)
return false

Statement Coverage:

Test case#	Input	Expected Output	Comments/Remarks
1	(0, null)	true	covers 1103,1104,1105,,1106,1109,1110, 1111,1112,1113,1114-1128
2	(0,4)	true	covers 1103,1104,1105,,1106,1109,1112, 1127
3	(0,12)	true	covers 1103-1111,1112

Test Result:

passesMillerRabinTest1() (0.006 s)
passesMillerRabinTest2() (0.001 s)
passesMillerRabinTest3() (0.001 s)
passesMillerRabinTest4() (0.008 s)

Branch Coverage:

Test case#	Input	Expected Output	Comments/Remarks
1	(0, null)	true	covers B1109(T), B1112(T), B1117(T), B1121(T)
2	(0, 4)	true	covers B1109(F), B1112(F)
4	(0,12)	true	covers B1109(T) B1112(T), B1117(T), B1121(T), B1122(T)

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Expected Output	Comments/Remarks

1	(0, null)	true	covers C1109(T), C1112(T), C1117(T), C1121(T)
2	(0, 4)	true	covers C1109(F), C1112(F)
3	(0,12)	true	covers C1109(T), C1112(T), C1117(T), C1121(T), C1122(T)

Boundary Interior:

Below we are taking line numbers to execute boundary interior.

1112 -> 1114

1112 -> 1114 -> 1115

1112 -> 1114 -> 1116 -> 1117

1112 -> 1114 -> 1116 -> 1117 -> 1116

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1123

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124 -> 1121

1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124 -> 1121 -> 1127

Test case#	Input	Expected Output	Comments/Remarks
1	(4, null)	True	Covers 1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124 -> 1121 -> 1126
2	(0, 4)	True	Covers 1112 -> 1114 -> 1116 -> 1117 -> 1116 -> 1119 -> 1120 -> 1121 -> 1122 -> 1124 -> 1121 -> 1127

Loop Boundary:

Test case#	Input	Expected Output	Comments/Remarks
1	(0,4)	True	Covers 1109T When the loop will not execute
2	(0,12)	True	Covers 1112T once
3	(4,2)	False	Covers 1112T more than one passes

Basis Path:

No of decision points = 3

No. of basis path = No of decision points +1 = 3+1 = 4

Path 1:

1101, 1103, 1104, 1105, 1106, 1127

Path 2:

1101, 1103, 1104, 1105, 1106, 1109, 1110, 1127

Path 3:

1101, 1103, 1104, 1105, 1106, 1109, 1110, 1112, 1113, 1114, 1115, 1116, 1117, 1119, 1120, 1127

Path 4:

1101, 1103, 1104, 1105, 1106, 1109, 1110, 1112, 1113, 1114, 1115, 1116, 1117, 1119, 1120, 1121, 1122, 1123, 1124, 1127

Test case#	Input	Expected Output	Comments/Remarks
1	(4, null)	True	Covers Path 1
2	(0, 4)	True	Covers Path 2
3	(null, null)	True	Covers Path 3
4	(7,9)	False	Covers Path 4

Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	iterations	1101	1112
2	Rnd	1101,1110	1109,1116
3	A	1105	1106

Variable #	Variable Name	DU pairs
1	iterations	<1101,1112>
2	Rnd	<1101,1109>,<1110,1116>
3	A	<1105,1106>

Test case#	Input	Expected Output	Comments/Remarks
1	(4, null)	True	For iterations: Not defined and used For Rnd: <1101,1109>,<1110,1116> For A: <1105,1106> It returns true second null value is handled in function
2	(7,9)	False	For iterations: <1101,1112> For Rnd: <1101,1109>,<1110,1116> For A: <1105,1106> It returns the result false due to its values

FUNCTION 10:

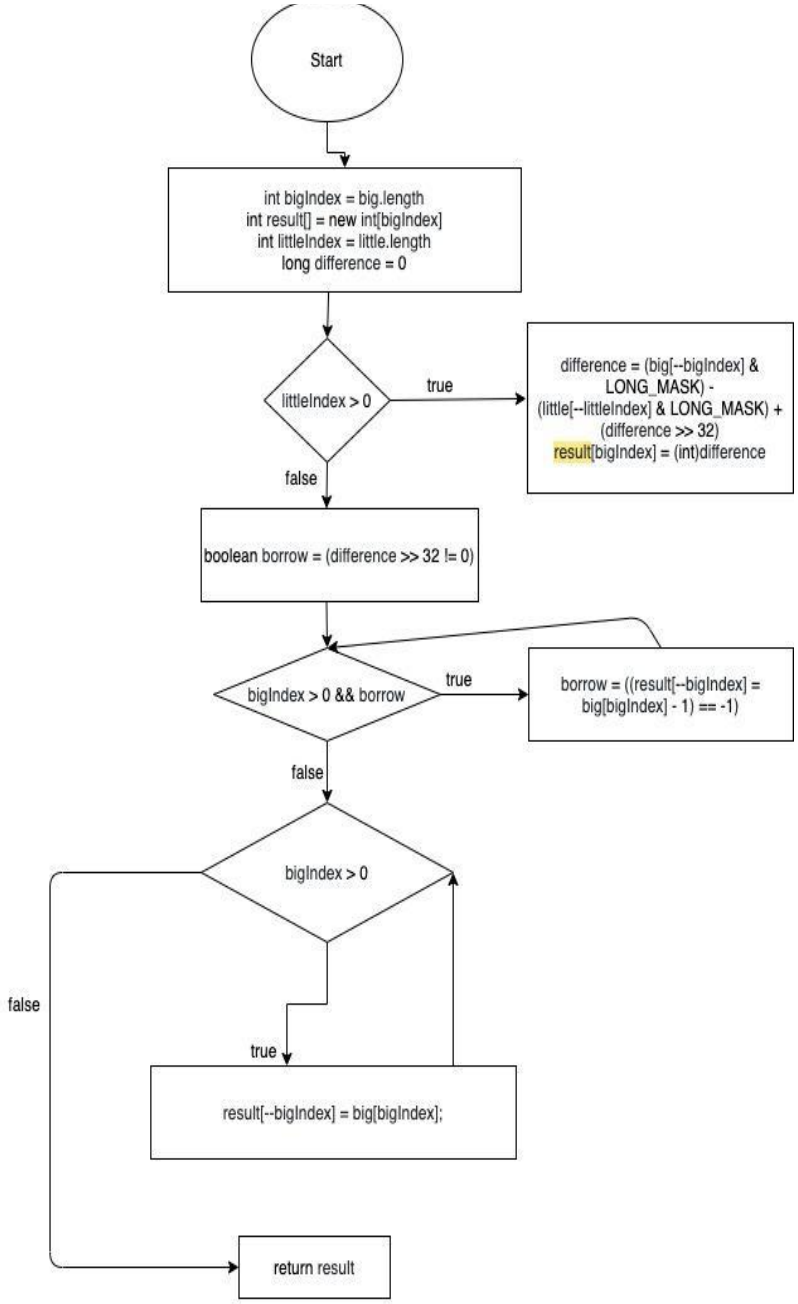
Source Code:

```java
    private static int[] subtract(int[] big, int[] little) {
        int bigIndex = big.length;
        int result[] = new int[bigIndex];
        int littleIndex = little.length;
        long difference = 0;

        // Subtract common parts of both numbers
        while (littleIndex > 0) {
            difference = (big[--bigIndex] & LONG_MASK) -
                         (little[--littleIndex] & LONG_MASK) +
                         (difference >> 32);
            result[bigIndex] = (int)difference;
        }

        // Subtract remainder of longer number while borrow propagates
        boolean borrow = (difference >> 32 != 0);
        while (bigIndex > 0 && borrow)
            borrow = ((result[--bigIndex] = big[bigIndex] - 1) == -1);

        // Copy remainder of longer number
        while (bigIndex > 0)
            result[--bigIndex] = big[bigIndex];

        return result;
    }

```

Test Case Code:

```java
@Test
void subtractTest1() {
    int[] intArray = {10,20};
    int[] blankArray = {};
    int[] resultArray = {30,40};
    assertArrayEquals(mu.subtract(10, intArray), resultArray);
}

@Test
void subtractTest2() {
    int[] intArray = {};
    int[] blankArray = {};
    int[] resultArray = {30,40};
    assertArrayEquals(mu.subtract(0, intArray), resultArray);
}

@Test
void subtractTest3() {
    int[] intArray = {10,20,30};
    int[] blankArray = {};
    int[] resultArray = {10,20};
    assertThrows(ArrayIndexOutOfBoundsException.class, () -> mu.subtract(12, blankArray), "Exception Raises");
}
```

CFG:


Start
int bigIndex = big.length
int result[] = new int[bigIndex]
int littleIndex = little.length
long difference = 0
littleIndex > 0
true
difference = (big[--bigIndex] & LONG_MASK) - (little[--littleIndex] & LONG_MASK) + (difference >> 32)
result[bigIndex] = (int)difference
false
boolean borrow = (difference >> 32 != 0)
bigIndex > 0 && borrow
true
borrow = ((result[--bigIndex] = big[bigIndex] - 1) == -1)
false
bigIndex > 0
false
true
result[--bigIndex] = big[bigIndex];
return result

Statement Coverage:

Test case#	Input	Expected Output	Comments/Remarks
1	x = {10,20} y = {30,40}	[30,40]	covers 1549, 1550, 1551, 1552, 1553, 1555, 1563,1564, 1565, 1568
2	x={10,20} y = {}	[10,20]	covers 1549, 1550, 1551, 1552, 1553, 1555, 1563,1564, 1565, 1568, 1569
3	x = {10,20,30} y = {10, 20}	Exception	raises exception

Branch Coverage:

Test case#	Input	Expected Output	Comments/Remarks
1	x = {10, 20} y = {30, 40}	[30,40]	covers B1555T, B1564T, B1568T
2	x = {10,20} y = {}	[10,20]	covers B1555F, B1564T, B1568T
3	x = {10,20,30} y = {10, 20}	Exception	covers B1555F, B1564F, B1568F

Test Result:

subtractTest1() (0.002 s)
subtractTest2() (0.001 s)
subtractTest3() (0.037 s)

Condition Coverage with Short Circuit Evaluation:

Test case#	Input	Expected Output	Comments/Remarks
1	x = {10,20}; y = {30,40}	[30,40]	covers C1555T, C1564T, C1568T
2	x={10,20} y = {}	[10,20]	covers C1555F, C1564T, C1568T
3	x = {10,20,30} y = {10, 20}	Exception	covers C1555F, C1564F, C1568F

Boundary Interior:

Loop 1:
1555 -> 1556
1555 -> 1556 -> 1557
1555 -> 1556 -> 1557 -> 1558
1555 -> 1556 -> 1557 -> 1558 -> 1559
1555 -> 1556 -> 1557 -> 1558 -> 1559 -> 1555
Loop 2:
1564 -> 1565
1564 -> 1565 - 1564
Loop 3:
1568 -> 1569
1568 -> 1569 -> 1568

Test case#	Input	Expected Output	Comments/Remarks

1	x = {10,20} y = {30,40}	[30,40]	Covers Loop 2 Covers Loop 1
2	x={10,20} y = {}	[10,20]	Covers Loop 2 Covers Loop 3

Loop Boundary:

Test case#	Input	Expected Output	Comments/Remarks
1	([0,2], [])	[0,2]	Covers: Loop 1: 1555T Loop 2: 1564T Loop 3: 1568T When the loop will not execute
2	([5],[2])	[2,4]	loop 1: 1555T loop 2: 1564T loop 3: 1568T Only one iteration
3	([10,20], [30,40])	[30,40]	loop 1: littleIndex > 0 True loop 2: bigIndex > 0 True loop 3: bigIndex > 0 True more than one passes

Basis Path:

No of decision points = 4

No. of basis path = No of decision points +1 = 4+1 = 5

Path 1:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1571

Path 2:

1548, 1549, 1550, 1551, 1552, 1563, 1564, 1565, 1571

Path 3:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1564, 1565, 1568, 1569, 1571

Path 4:

1548, 1549, 1550, 1551, 1552, 1555, 1556, 1557, 1558, 1559, 1563, 1568, 1569, 1571

Test case#	Input	Expected Output	Comments/Remarks
1	x = {10,20} y = {30,40}	[30,40]	Covers Path 3
2	x={10,20} y = {}	[10,20]	Covers Path 2
3	x={} y = {10,20}	Exception	Covers Path 1
4	x = {10,20,30} y = {10, 20}	Exception	Covers Path 4

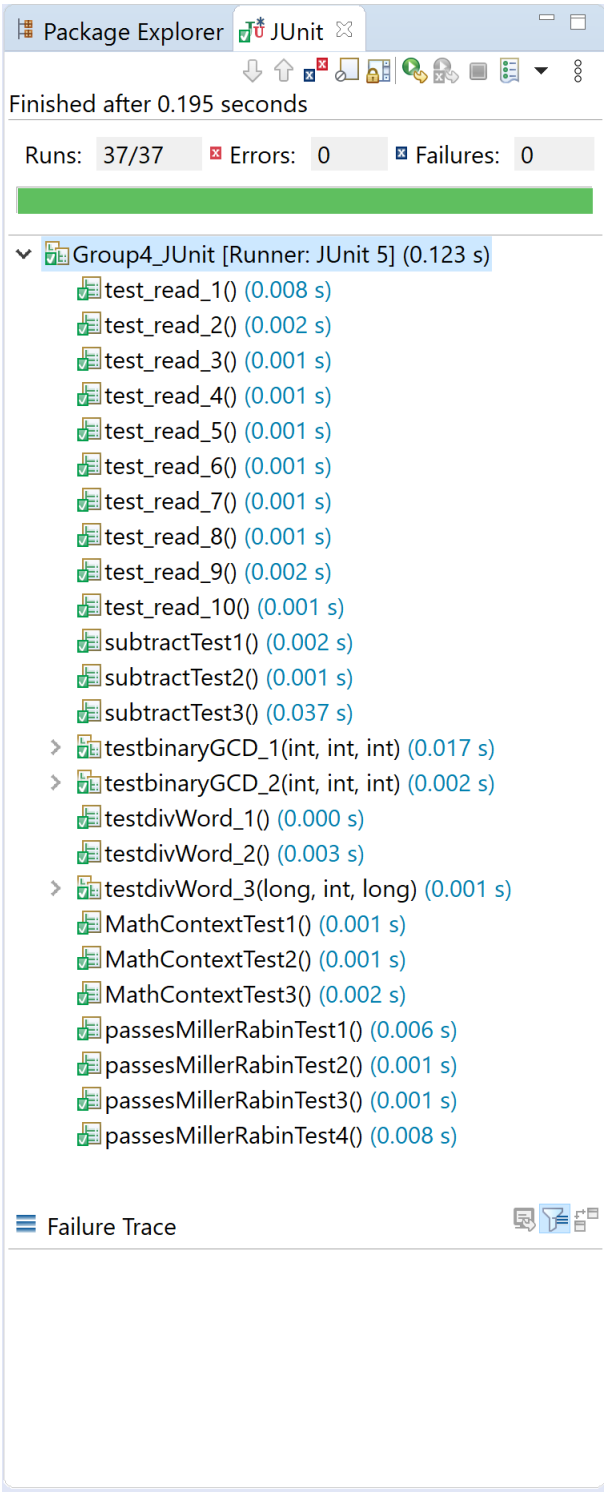
Data Flow Testing:

Variable #	Variable Name	Definitions	Uses
1	Big	1548	1549, 1556, 1565, 1569
2	Little	1548	1551,1556
3	Borrow	1563,1565	1564

Variable #	Variable Name	DU pairs
1	Big	<1548,1549>,<1548,1549><1548,1556><1565,1569>
2	Little	<1548,1551>,<1548,1556>
3	Borrow	<1563,1564>

Test case#	Input	Output	Expected Output	Pass/Fail	Comments/Remarks
1	x = {10, 20} y = {30, 40}	[30,40]	[-21,20]	Pass	For big covers <1548,1549>, <1565,1569> For little covers <1548,1551>, <1548,1556> For borrow covers <1563,1564> It returns true second null value is handled in function
2	x={10,20} y = {}	[10,20]	[10,20]	Pass	For big covers <1548,1549>, <1565,1569> For little covers <1548,1551>, <1548,1556> For borrow covers <1563,1564> It returns the result false due to its values

All Test Cases Result:



```xml
<?xml version="1.0" encoding="UTF-8"?>
- <testrun ignored="0" errors="0" failures="0" started="37" tests="37" project="JunitAssignment" name="Group4_JUnit">
  - <testsuite name="app.code.Group4_JUnit" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]" displayname="Group4_JUnit" time="0.191">
      <testcase name="test_read_1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_1()]" displayname="test_read_1()" time="0.024" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_2()]" displayname="test_read_2()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_3()]" displayname="test_read_3()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_4" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_4()]" displayname="test_read_4()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_5" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_5()]" displayname="test_read_5()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_6" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_6()]" displayname="test_read_6()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_7" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_7()]" displayname="test_read_7()" time="0.008" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_8" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_8()]" displayname="test_read_8()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_9" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_9()]" displayname="test_read_9()" time="0.002" classname="app.code.Group4_JUnit"/>
      <testcase name="test_read_10" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:test_read_10()]" displayname="test_read_10()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="subtractTest1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:subtractTest1()]" displayname="subtractTest1()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="subtractTest2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:subtractTest2()]" displayname="subtractTest2()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="subtractTest3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:subtractTest3()]" displayname="subtractTest3()" time="0.061" classname="app.code.Group4_JUnit"/>
    - <testsuite name="testbinaryGCD_1(app.code.Group4_JUnit)" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_1(int, int, int)]" displayname="testbinaryGCD_1(int, int, int)" time="0.03" parameters="int,int,int">
        <testcase name="testbinaryGCD_1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_1(int, int, int)]/[test-template-invocation:#1]" displayname="[1] 15, 0, 15" time="0.03" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_1(int, int, int)]/[test-template-invocation:#2]" displayname="[2] 0, 15, 15" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_1(int, int, int)]/[test-template-invocation:#3]" displayname="[3] 98, 56, 14" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_1(int, int, int)]/[test-template-invocation:#4]" displayname="[4] 56, 98, 14" time="0.01" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
      </testsuite>
    - <testsuite name="testbinaryGCD_2(app.code.Group4_JUnit)" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]" displayname="testbinaryGCD_2(int, int, int)" time="0.0" parameters="int,int,int">
        <testcase name="testbinaryGCD_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]/[test-template-invocation:#1]" displayname="[1] 12, 12, 12" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]/[test-template-invocation:#2]" displayname="[2] 4, 2, 2" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]/[test-template-invocation:#3]" displayname="[3] 6, 2, 2" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]/[test-template-invocation:#4]" displayname="[4] 10, 2, 2" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]/[test-template-invocation:#5]" displayname="[5] 12, 2, 2" time="0.008" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
        <testcase name="testbinaryGCD_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testbinaryGCD_2(int, int, int)]/[test-template-invocation:#6]" displayname="[6] 14, 2, 2" time="0.0" classname="app.code.Group4_JUnit"
            parameters="int,int,int" dynamicTest="true"/>
      </testsuite>
      <testcase name="testdivWord_1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:testdivWord_1()]" displayname="testdivWord_1()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="testdivWord_2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:testdivWord_2()]" displayname="testdivWord_2()" time="0.008" classname="app.code.Group4_JUnit"/>
    - <testsuite name="testdivWord_3(app.code.Group4_JUnit)" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testdivWord_3(long, int, long)]" displayname="testdivWord_3(long, int, long)" time="0.0" parameters="long,int,long">
        <testcase name="testdivWord_3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testdivWord_3(long, int, long)]/[test-template-invocation:#1]" displayname="[1] 10, 5, 2" time="0.0" classname="app.code.Group4_JUnit"
            parameters="long,int,long" dynamicTest="true"/>
        <testcase name="testdivWord_3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testdivWord_3(long, int, long)]/[test-template-invocation:#2]" displayname="[2] 5, 3, 8589934593" time="0.0" classname="app.code.Group4_JUnit"
            parameters="long,int,long" dynamicTest="true"/>
        <testcase name="testdivWord_3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testdivWord_3(long, int, long)]/[test-template-invocation:#3]" displayname="[3] 20, 3, 8589934598" time="0.0" classname="app.code.Group4_JUnit"
            parameters="long,int,long" dynamicTest="true"/>
        <testcase name="testdivWord_3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testdivWord_3(long, int, long)]/[test-template-invocation:#4]" displayname="[4] 28, 3, 4294967305" time="0.008" classname="app.code.Group4_JUnit"
            parameters="long,int,long" dynamicTest="true"/>
        <testcase name="testdivWord_3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[test-template:testdivWord_3(long, int, long)]/[test-template-invocation:#5]" displayname="[5] 32, 3, 8589934602" time="0.0" classname="app.code.Group4_JUnit"
            parameters="long,int,long" dynamicTest="true"/>
      </testsuite>
      <testcase name="MathContextTest1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:MathContextTest1()]" displayname="MathContextTest1()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="MathContextTest2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:MathContextTest2()]" displayname="MathContextTest2()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="MathContextTest3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:MathContextTest3()]" displayname="MathContextTest3()" time="0.008" classname="app.code.Group4_JUnit"/>
      <testcase name="passesMillerRabinTest1" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:passesMillerRabinTest1()]" displayname="passesMillerRabinTest1()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="passesMillerRabinTest2" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:passesMillerRabinTest2()]" displayname="passesMillerRabinTest2()" time="0.008" classname="app.code.Group4_JUnit"/>
      <testcase name="passesMillerRabinTest3" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:passesMillerRabinTest3()]" displayname="passesMillerRabinTest3()" time="0.0" classname="app.code.Group4_JUnit"/>
      <testcase name="passesMillerRabinTest4" uniqueid="[engine:junit-jupiter]/[class:app.code.Group4_JUnit]/[method:passesMillerRabinTest4()]" displayname="passesMillerRabinTest4()" time="0.008" classname="app.code.Group4_JUnit"/>
    </testsuite>
  </testrun>
```