TASK -1

1) A.

Method 1: BigInteger(int signum, byte[] magnitude)

API Specification (Appendix, Page 4):

- Translates a sign-magnitude representation into a BigInteger.
- Parameters:
  - signum: -1 (negative), 0 (zero), or 1 (positive).
  - magnitude: Big-endian byte array (most significant byte at index 0). A zero-length array is permissible and results in a value of 0 (regardless of signum).
- Throws:
  - NumberFormatException if signum is not -1, 0, or 1, or if signum is 0 and magnitude contains non-zero bytes.

Equivalence Classes:

- **Signum**:
  - Valid: signum = -1, 0, 1.
  - Invalid: signum < -1 (e.g., -2), signum > 1 (e.g., 2).
- **Magnitude**:
  - Zero-length array: magnitude.length == 0.
  - Non-zero length with all zeros: e.g., magnitude = {0, 0}.
  - Non-zero length with non-zero bytes: e.g., magnitude = {1, 2}.
  - Invalid when signum == 0: magnitude contains non-zero bytes (e.g., {1, 2}).
- **Boundary Cases**:
  - magnitude with leading zeros (e.g., {0, 0, 1} should be equivalent to {1}).
  - signum == 0 with zero magnitude.

- **Test Cases**:

ID	**Partition Tested**	**Test Inputs**	**Expected Output**
1	Valid signum, zero-length magnitude	signum = 1, magnitude = {}	BigInteger value of "0"
2	Valid signum, non-zero magnitude (positive)	signum = 1, magnitude = {1, 2}	BigInteger value of "258"
3	Valid signum, non-zero magnitude (negative)	signum = -1, magnitude = {1, 2}	BigInteger value of "-258"
4	signum == 0, all-zero magnitude	signum = 0, magnitude = {0, 0}	BigInteger value of "0"
5	Invalid signum	signum = 2, magnitude = {1, 2}	Throws NumberFormatException
6	signum == 0, non-zero magnitude (invalid)	signum = 0, magnitude = {1, 2}	Throws NumberFormatException

Method 2: BigInteger(String val, int radix)

API Specification (Appendix, Page 4):

- Translates a string representation in the specified radix into a BigInteger.
- Parameters:
  - val: String representation (optional minus/plus sign followed by digits in the given radix).
  - radix: Base to interpret val (must be between Character.MIN_RADIX (2) and Character.MAX_RADIX (36)).
- Throws:
  - NumberFormatException if val is not a valid representation in the specified radix, or if radix is outside the valid range.

Equivalence Classes:

- **Radix**:
  - Valid: radix in [Character.MIN_RADIX, Character.MAX_RADIX] (2 to 36).
  - Invalid: radix < Character.MIN_RADIX (e.g., 1), radix > Character.MAX_RADIX (e.g., 37).
- **String (val)**:
  - Valid positive: e.g., "123" for radix 10.
  - Valid negative: e.g., "-123" for radix 10.
  - Valid with hex digits: e.g., "FF" for radix 16.

- Invalid digits: e.g., "12G" for radix 10 (G is not a valid digit in base 10).
- Invalid format: e.g., "12 3" (contains whitespace).
- Zero: "0".

- **Boundary Cases**:
  - radix = 2 (min), radix = 36 (max).
  - Empty string (though not explicitly tested here, as it's not a valid input per the API).
- **Test Cases**:

ID	Partition Tested	Test Inputs	Expected Output
1	Valid radix, valid positive string	val = "123", radix = 10	BigInteger value of "123"
2	Valid radix, valid negative string	val = "-123", radix = 10	BigInteger value of "-123"
3	Valid radix = 16, valid hex string	val = "FF", radix = 16	BigInteger value of "255"
4	Invalid radix	val = "123", radix = 1	Throws NumberFormatException
5	Valid radix, invalid digits for radix	val = "12G", radix = 10	Throws NumberFormatException
6	Valid radix, invalid string format	val = "12 3", radix = 10	Throws NumberFormatException

Method 3: compareTo(BigInteger val)

API Specification (Appendix, Page 4):

- Compares this BigInteger with the specified val.
- Parameters:
  - val: BigInteger to compare against.
- Returns:
  - -1 if this < val, 0 if this == val, 1 if this > val.

Equivalence Classes:

- **Comparison Outcome**:
  - This < val: e.g., this = 5, val = 10.
  - This == val: e.g., this = 5, val = 5.
  - This > val: e.g., this = 10, val = 5.
- **Sign**:
  - Both positive.
  - Both negative.

- This positive, val negative.
- This negative, val positive.

- **Size**:
  - Small numbers (fit in ival).
  - Large numbers (require words array).
- **Boundary Cases**:
  - Zero comparison: this = 0, val = 0.
- **Test Cases**:

ID	Partition Tested	Test Inputs	Expected Output
1	This < val, both positive, small numbers	this = "5", val = "10"	-1
2	This == val, both zero	this = "0", val = "0"	0
3	This > val, different signs	this = "5", val = "-5"	1
4	This < val, both large numbers	this = "1234567890", val = "12345678901234567890"	-1
5	This > val, both negative	this = "-5", val = "-10"	1

B)

Implement and run the test cases in JUnit

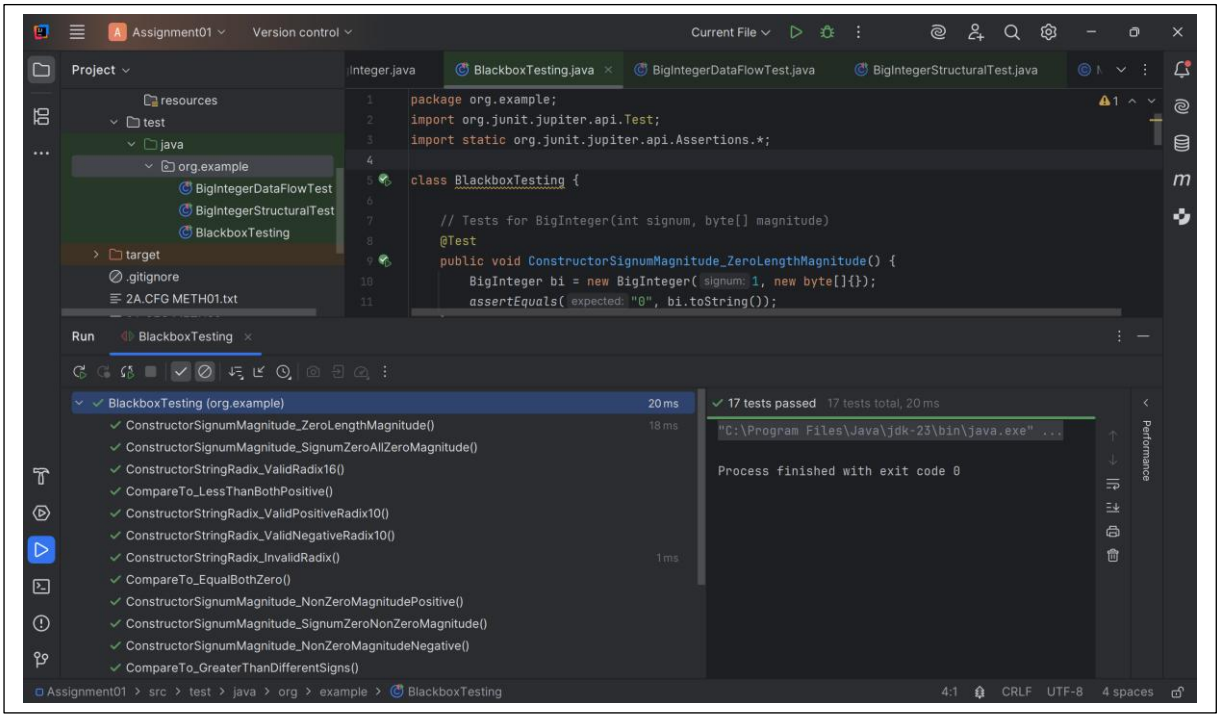
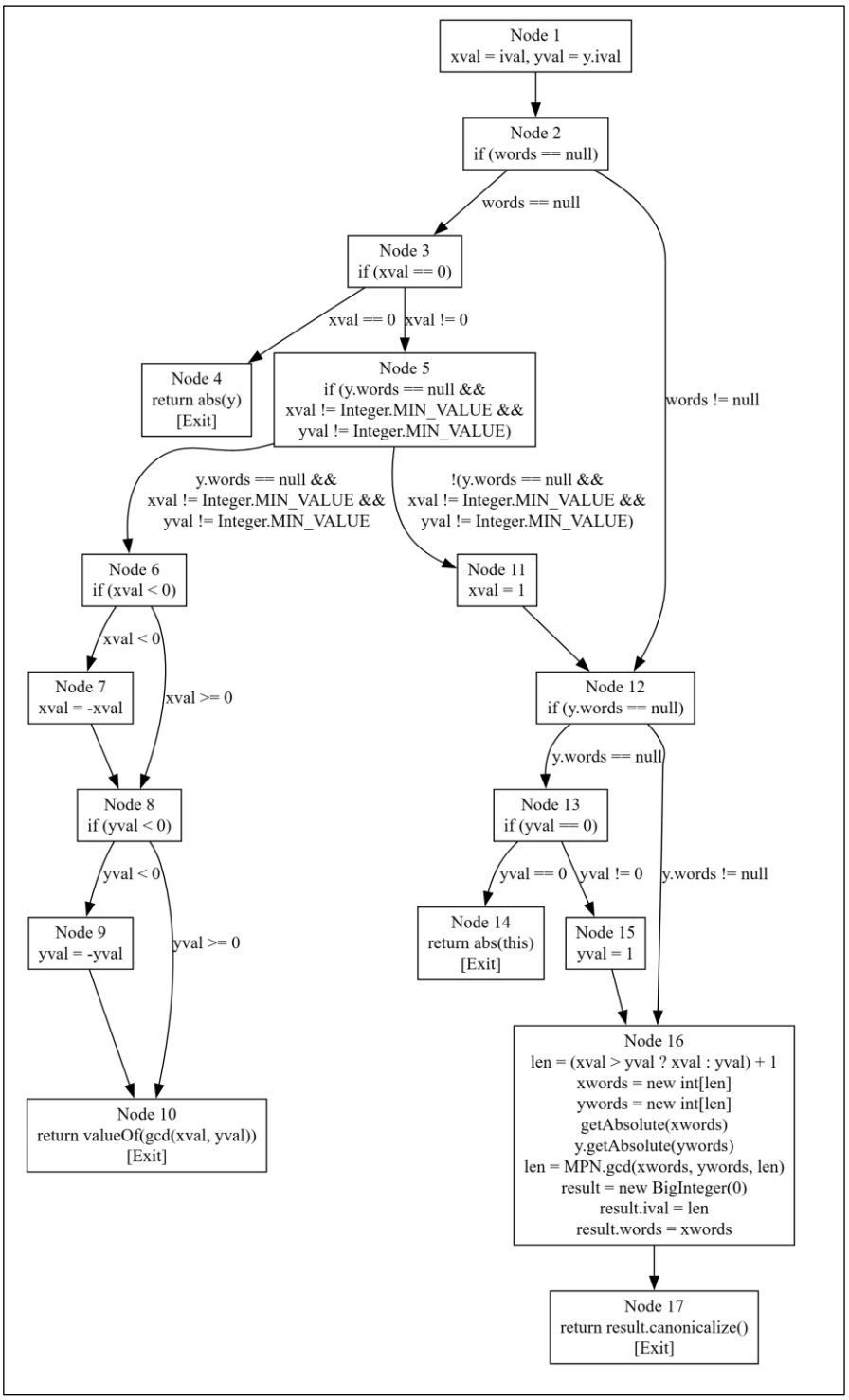


Fig: Implement of Blackbox testing

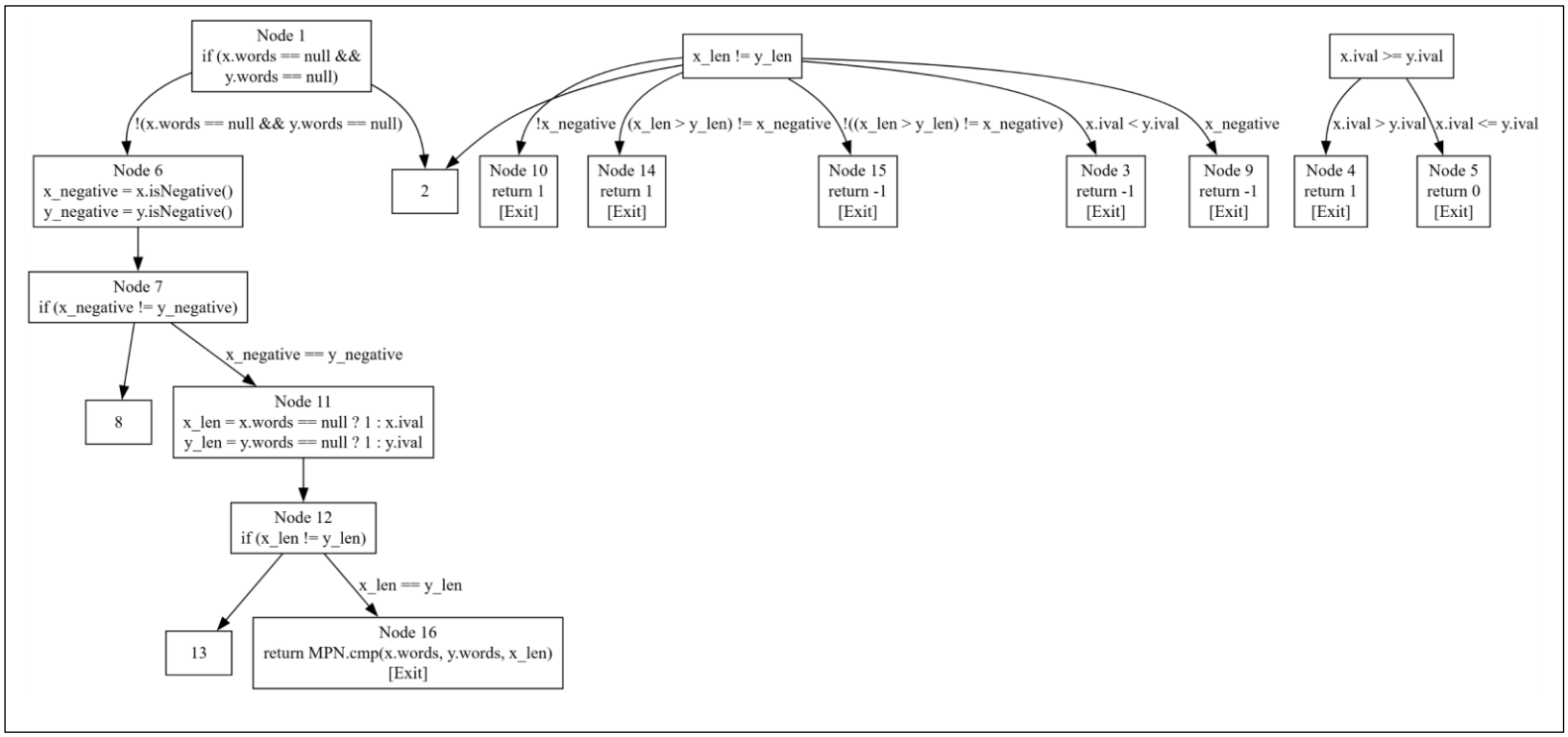
TASK 02- Whitebox Testing: Structural Testing

A)

- **CFG for public BigInteger gcd(BigInteger y)**
  The `gcd(BigInteger y)` method computes the greatest common divisor (GCD) of two `BigInteger` objects (`this` and `y`). It first checks if the numbers can be handled using their `ival` fields (if `words` is null and other conditions are met), computes the GCD of the `ival` values if possible, and otherwise computes the GCD using the `words` arrays via the `MPN.gcd` method. The control flow graph for `gcd(BigInteger y)` is shown below.



- **CFG for private static int compareTo(BigInteger x, BigInteger y)**
  The `compareTo(BigInteger x, BigInteger y)` method compares two `BigInteger` objects (`x` and `y`) and returns -1 if `x < y`, 0 if `x == y`, or 1 if `x > y`. It first compares the `ival` fields if both numbers are small (i.e., `words` is null), then compares signs, lengths, and finally the `words` arrays if necessary. The control flow graph for `compareTo(BigInteger x, BigInteger y)` is shown below.



B) Test cases to achieve 100% statement coverage for the “public BigInteger gcd(BigInteger y)” method.

ID	Test Inputs	Expected Output	Nodes Covered
1	this.words = null, this.ival = 0, y.words = null, y.ival = 10	BigInteger("10")	1, 2, 3, 4
2	this.words = null, this.ival = -4, y.words = null, y.ival = -6	BigInteger("2")	1, 2, 3, 5, 6, 7, 8, 9, 10
3	this.words = null, this.ival = Integer.MIN_VALUE, y.words = null, y.ival = 0	BigInteger("2147483648")	1, 2, 3, 5, 11, 12, 13, 14
4	this.words = {1}, this.ival = 2, y.words = null, y.ival = 3	BigInteger("1")	1, 2, 12, 13, 15, 16, 17
5	this.words = {15}, this.ival = 2, y.words = {25}, y.ival = 2	BigInteger("5")	1, 2, 12, 16, 17

- Test cases to achieve 100% statement coverage for the “private static int compareTo(BigInteger x, BigInteger y)” method-

ID	Inputs	Expected Output	Nodes Covered
1	x.words = null, x.ival = 3 y.words = null, y.ival = 5	-1	1 → 2 → 3
2	x.words = null, x.ival = 5 y.words = null, y.ival = 3	1	1 → 2 → 4
3	x.words = null, x.ival = 5 y.words = null, y.ival = 5	0	1 → 2 → 5
4	x.words = {}, x.ival = 2 y.words = null, y.ival = 1	1	1 → 6 → 7 → 8 → 11 → 12 → 14
5	x.words = {}, x.ival = 1 y.words = {}, y.ival = 2	-1	1 → 6 → 7 → 8 → 11 → 12 → 15
6	x.words = {-1}, x.ival = 1 y.words = {1}, y.ival = 1	-1	1 → 6 → 7 → 9
7	x.words = {-1, -1}, x.ival = 2 y.words = {-1, -2}, y.ival = 2	1	1 → 6 → 7 → 8 → 11 → 12 → 13 → 16

C) Method: gcd(BigInteger y) – Branch Coverage Test Cases

ID	Inputs	Expected Output	Branches Covered
1	this.words = null, this.ival = 0 y.words = null, y.ival = 10	BigInteger("10")	if this.words == null is **true**
2	this.words = null, this.ival = -4 y.words = null, y.ival = -6	BigInteger("2")	if (y.words == null) and both ival ≠ MIN
3	this.words = null, this.ival = Integer.MIN_VALUE y.ival = 0	BigInteger("2147483648")	else branch of y.words == null
4	this.words = {15}, this.ival = 2 y.words = {25}, y.ival = 2	BigInteger("5")	else branch of outermost condition

- Method: compareTo(BigInteger x, BigInteger y) – Branch Coverage Test Cases

ID	Inputs	Expected Output	Branches Covered
1	x.words = null, x.ival = 5 y.words = null, y.ival = 10	-1	if (x.words == null && y.words == null) → true
2	x.words = {-1}, x.ival = 1 y.words = {1}, y.ival = 1	-1	if (x_negative != y_negative) → true
3	x.words = {1}, x.ival = 2 y.words = {1}, y.ival = 1	1	if (x_len != y_len) → true
4	x.words = {1}, x.ival = 2 y.words = {2}, y.ival = 2	-1	return MPN.cmp(...) case

D) Method: gcd(BigInteger y) – Test cases to achieve 100% condition coverage

Conditions	Inputs	Expected Output	Conditions Covered
words == null	(0, 42)	42	words == null (true), xval == 0 (true)
xval == 0	(12, 18)	6	words == null (true), xval == 0 (false), y.words == null (true), xval != MIN_VALUE (true), yval != MIN_VALUE (true), xval < 0 (false), yval < 0 (false)
y.words == null	(-12, -18)	6	xval < 0 (true), yval < 0 (true)
xval != Integer.MIN_VALUE	(Integer.MIN_VALUE, 18)	Implementation-specific	words == null (true), y.words == null (true), xval != MIN_VALUE (false)
yval != Integer.MIN_VALUE	(12, Integer.MIN_VALUE)	Implementation-specific	words == null (true), y.words == null (true), yval != MIN_VALUE (false)
xval < 0	(12, new BigInteger("12345678901234567890"))	Implementation-specific	words == null (true), y.words == null (false)
yval < 0	(new BigInteger("12345678901234567890"), 0)	12345678901234567890	words == null (false), y.words == null (true), yval == 0 (true)
y.words == null	(new BigInteger("12345678901234567890"), 1)	Implementation-specific	words == null (false), y.words == null (true), yval == 0 (false)
yval == 0	(new BigInteger("12345678901234567890"), new BigInteger("98765432109876543210"))	Implementation-specific	words == null (false), y.words == null (false)

- Method: compareTo(BigInteger x, BigInteger y) – test cases to achieve 100% condition coverage

Conditions	Inputs	Expected Output	Conditions Covered
x.words == null	(1, 2)	-1	x.words == null (true), y.words == null (true), x.ival < y.ival (true)
y.words == null	(2, 1)	1	x.words == null (true), y.words == null (true), x.ival > y.ival (true)
x.ival < y.ival	(1, 1)	0	x.words == null (true), y.words == null (true), x.ival == y.ival (true)
x.ival > y.ival	(-1, 1)	-1	x.words == null (true/false), y.words == null (true/false), x_negative != y_negative (true), x_negative (true)
x_negative	(1, -1)	1	x.words == null (true/false), y.words == null (true/false), x_negative != y_negative (true), x_negative (false)
y_negative	(new BigInteger("12345678901234567890"), 1)	1	x.words == null (false), y.words == null (true), x_negative != y_negative (false), x_len != y_len (true), (x_len > y_len) != x_negative (true)
x_negative != y_negative	(new BigInteger("-12345678901234567890"), -1)	-1	x.words == null (false), y.words == null (true), x_negative != y_negative (false), x_len != y_len (true), (x_len > y_len) != x_negative (false)
x_len != y_len	(new BigInteger("12345678901234567890"), new BigInteger("12345678901234567890"))	0	x.words == null (false), y.words == null (false), x_negative != y_negative (false), x_len != y_len (false)
x_len > y_len	(new BigInteger("12345678901234567890"), new BigInteger("12345678901234567891"))	-1	x.words == null (false), y.words == null (false), x_negative != y_negative (false), x_len != y_len (false)

E) Method: gcd(BigInteger y) – test cases to achieve 100% condition/decision coverage

Conditions	Inputs	Expected Output	Conditions/Decisions Covered
words == null	(0, 42)	42	words==null (T), xval==0 (T)
xval == 0	(12, 18)	6	words==null (T), xval==0 (F), y.words==null (T), xval!=MIN_VALUE (T), yval!=MIN_VALUE (T), xval<0 (F), yval<0 (F)
y.words == null && xval != Integer.MIN_VALUE && yval != Integer.MIN_VALUE	(-12, -18)	6	xval<0 (T), yval<0 (T)
xval < 0	(Integer.MIN_VALUE, 18)	-1	xval!=MIN_VALUE (F)
yval < 0	(12, new BigInteger("12345678901234567890"))	1	y.words==null (F)
y.words == null	(new BigInteger("12345678901234567890"), 1)	12345678901234567890	words==null (F), yval==0 (T)
yval == 0	(new BigInteger("-12345678901234567890"), -1)	1	words==null (F), y.words==null (T), yval==0 (F)

Method: compareTo(BigInteger x, BigInteger y) – test cases to achieve 100% condition/decision coverage

Conditions	Inputs	Expected Output	Branches Covered
x.words == null && y.words == null	(1, 2)	-1	x.words&&y.words==null (T), x.ival<y.ival (T)
x.ival < y.ival	(2, 1)	1	x.words&&y.words==null (T), x.ival>y.ival (T)
x.ival > y.ival	(1, 1)	0	x.words&&y.words==null (T), x.ival==y.ival (T)
x_negative != y_negative	(-1, 1)	-1	x_negative!=y_negative (T), x_negative (T)
x_len != y_len	(1, -1)	1	x_negative!=y_negative (T), x_negative (F)
(x_len > y_len) != x_negative	(new BigInteger("100"), 1)	1	x_len>y_len (T), (x_len>y_len)!=x_negative (T)

F) Method: gcd(BigInteger y) – test cases to achieve 100% multiple condition coverage

Target Conditions: y.words == null && xval != MIN_VALUE && yval != MIN_VALUE

gcd(BigInteger y) Method

Conditions	y.words==null	xval!=MIN_VALUE	yval!=MIN_VALUE	Test Input (x, y)	Expected Output
1	T	T	T	(12, 18)	6
2	T	T	F	(12, Integer.MIN_VALUE)	
3	T	F	T	(Integer.MIN_VALUE, 18)	
4	T	F	F	(Integer.MIN_VALUE, Integer.MIN_VALUE)	
5	F	*	*	(12, new BigInteger("1e20"))	

compareTo(BigInteger x, BigInteger y) – test cases to achieve 100% multiple condition coverage

Condition 1: x.words == null && y.words == null

Conditions	x.words==null	y.words==null	Test Input (x, y)	Expected Output
1	T	T	(1, 2)	-1
2	T	F	(1, new BigInteger("1e20"))	-1
3	F	T	(new BigInteger("1e20"), 1)	1
4	F	F	(new BigInteger("1e20"), new BigInteger("2e20"))	-1

Condition 2: (x_len > y_len) != x_negative

Conditions	x_len>y_len	x_negative	!= Result	Test Input (x, y)	Expected Output
1	T	T	T	(new BigInteger("100"), 1)	1
2	T	F	F	(new BigInteger("-100"), -1)	-1
3	F	T	F	(new BigInteger("1"), 100)	-1
4	F	F	T	(new BigInteger("-1"), -100)	1

G) Method: gcd(BigInteger y) – test cases to achieve 100% modified condition/decision coverage

Target Decision: if (y.words == null && xval != MIN_VALUE && yval != MIN_VALUE)

Conditions pair	Input(x,y)	Changed condition
SimpleGCD → ComplexGCD	(12, 18) → (12, new BigInteger("1e20"))	y.words: T→F
ValidX → InvalidX	(12, 18) → (Integer.MIN_VALUE, 18)	xval≠MIN: T→F
ValidY → InvalidY	(12, 18) → (12, Integer.MIN_VALUE)	yval≠MIN: T→F

Decision 1: if (x.words == null && y.words == null)

Conditions pair	Input(x,y)	Changed condition
BothSimple → YComplex	(1, 2) → (1, new BigInteger("1e20"))	y.words: T→F

Decision 2: if (x_negative != y_negative)

Conditions pair	Input(x,y)	Changed condition
XNegative → YNegative	(-1, 1) → (1, -1)	x_negative: T→F

Decision 3: if (x_len != y_len)

Conditions pair	Input(x,y)	Changed condition
XLonger → YLonger	(new BigInteger("100"), 1) → (new BigInteger("1"), 100)	x_negative: x_len>y_len: T→F

H) Implement and run the test cases in JUnit.

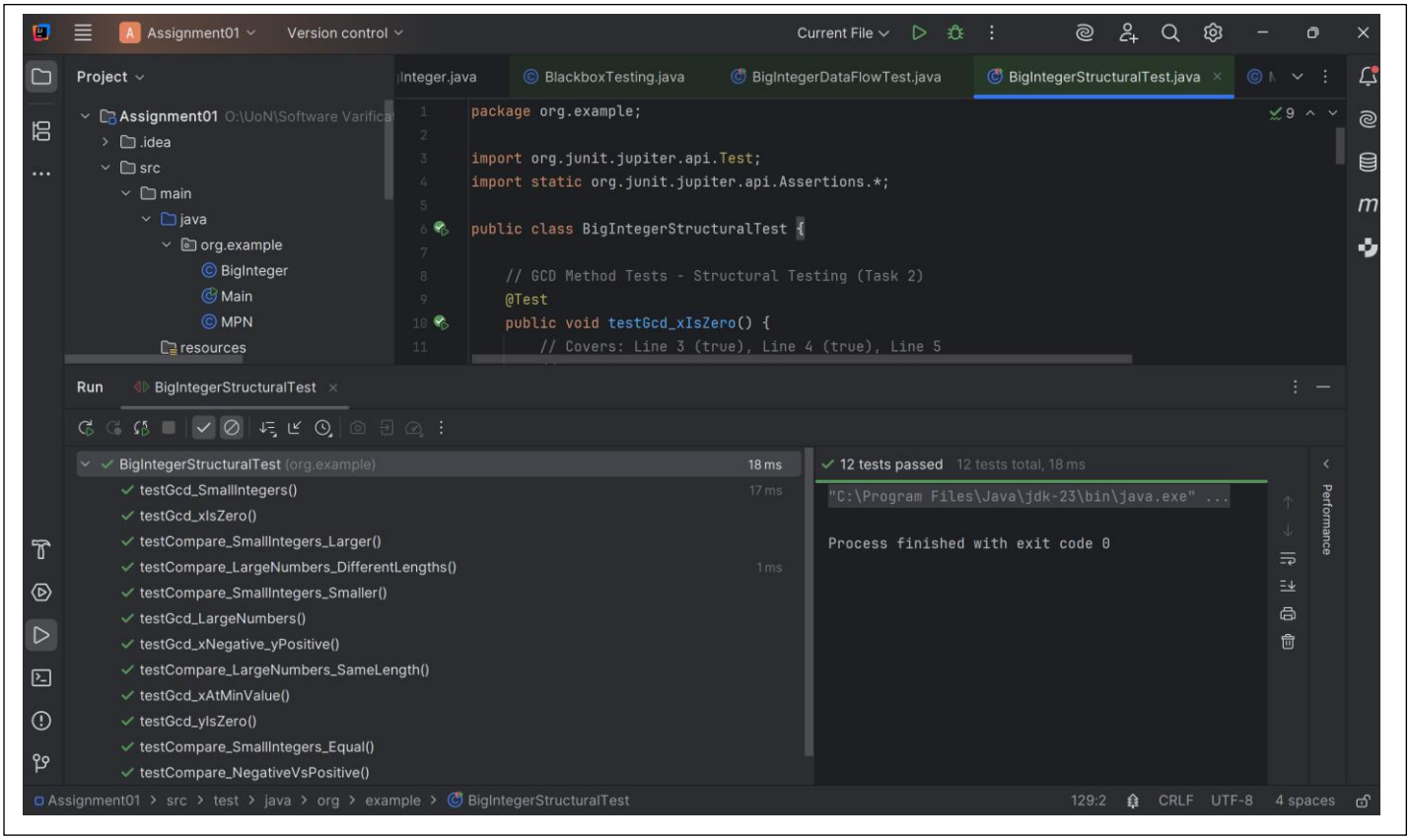


Fig: Implement of Structural testing

TASK 03- All Definition-Use (DU) Pairs

A) Identify All Definition-Use (DU) Pairs

Method: compareTo(BigInteger x, BigInteger y) *(accessed via public int compareTo(BigInteger val))*
This method compares two BigInteger objects (x and y) and returns -1, 0, or 1 based on their numerical order. It includes multiple conditional branches to handle sign comparison, length comparison, and magnitude comparison via the MPN.cmp method.

Notes:

- **c-use** = computational use (in calculations or expressions)
- **p-use** = predicate use (in conditions or decisions)

Variable	Definition Location	Use Location	Use Type	du-Pair Description
x_negative	x_negative = x.isNegative() (Line 3)	if (x_negative != y_negative) (Line 5)	p-use	Definition at line 3 used in condition
x_negative	x_negative = x.isNegative() (Line 3)	x_negative ? -1 : 1 (Line 6)	c-use	Definition at line 3 used in return
x_negative	x_negative = x.isNegative() (Line 3	(x_len > y_len) != x_negative (Line 10	c-use	Definition at line 3 used in return
y_negative	y_negative = y.isNegative() (Line 4)	if (x_negative != y_negative) (Line 5)	p-use	Definition at line 4 used in condition
x_len	x_len = x.words == null ? 1 : x.ival (Line 7)	if (x_len != y_len) (Line 9)	p-use	Definition at line 7 used in condition
x_len	x_len = x.words == null ? 1 : x.ival (Line 7)	(x_len > y_len) != x_negative (Line 10)	c-use	Definition at line 7 used in return
x_len	x_len = x.words == null ? 1 : x.ival (Line 7)	MPN.cmp(x.words, y.words, x_len) (Line 11)	c-use	Definition at line 7 used in comparison

y_len	y_len = y.words == null ? 1 : y.ival (Line 8)	if (x_len != y_len) (Line 9)	p-use	Definition at line 8 used in condition
y_len	y_len = y.words == null ? 1 : y.ival (Line 8)	(x_len > y_len) != x_negative (Line 10)	c-use	Definition at line 8 used in return

- method- public BigInteger gcd(BigInteger y)

Variable	Definition Location	Use Location	Use Type	du-Pair Description
xval	xval = ival (Line 1)	if (xval == 0) (Line 4)	p-use	Definition at line 1 used in condition
xval	xval = ival (Line 1)	xval != Integer.MIN_VALUE (Line 6)	p-use	Definition at line 1 used in condition
xval	xval = ival (Line 1)	if (xval < 0) (Line 7)	p-use	Definition at line 1 used in condition (if not killed)
xval	xval = ival (Line 1)	len = (xval > yval ? xval : yval) + 1 (Line 17)	p-use	Definition at line 1 used in expression (if not killed)
xval	xval = -xval (Line 8)	gcd(xval, yval) (Line 11)	c-use	Definition at line 8 used in GCD computation
xval	xval = -xval (Line 8)	len = (xval > yval ? xval : yval) + 1 (Line 17)	c-use	Definition at line 8 used in expression (if not killed)
xval	xval = 1 (Line 12)	len = (xval > yval ? xval : yval) + 1 (Line 17)	c-use	Definition at line 12 used in expression
yval	yval = y.ival (Line 2)	yval != Integer.MIN_VALUE (Line 6)	c-use	Definition at line 2 used in condition
yval	yval = y.ival (Line 2)	if (yval < 0) (Line 9)	p-use	Definition at line 2 used in condition (if not killed)

yval	yval = y.ival (Line 2)	if (yval == 0) (Line 14)	p-use	Definition at line 2 used in condition (if not killed)
yval	yval = y.ival (Line 2)	len = (xval > yval ? xval : yval) + 1 (Line 17)	p-use	Definition at line 2 used in expression (if not killed)
yval	yval = -yval (Line 10)	gcd(xval, yval) (Line 11)	c-use	Definition at line 10 used in GCD computation
yval	yval = -yval (Line 10)	len = (xval > yval ? xval : yval) + 1 (Line 17)	c-use	Definition at line 10 used in expression (if not killed)
yval	yval = 1 (Line 16)	len = (xval > yval ? xval : yval) + 1 (Line 17)	c-use	Definition at line 16 used in expression
len	len = (xval > yval ? xval : yval) + 1 (Line 17)	xwords = new int[len] (Line 18)	c-use	Definition at line 17 used in array creation
len	len = (xval > yval ? xval : yval) + 1 (Line 17)	ywords = new int[len] (Line 19)	c-use	Definition at line 17 used in array creation
len	len = (xval > yval ? xval : yval) + 1 (Line 17)	MPN.gcd(xwords, ywords, len) (Line 22)	c-use	Definition at line 17 used in GCD computation
len	len = MPN.gcd(xwords, ywords, len) (Line 22)	result.ival = len (Line 24)	c-use	Definition at line 22 used in assignment
xwords	xwords = new int[len] (Line 18)	getAbsolute(xwords) (Line 20)	c-use	Definition at line 18 used in method call

xwords	xwords = new int[len] (Line 18)	MPN.gcd(xwords, ywords, len) (Line 22)	c-use	Definition at line 18 used in GCD computation
xwords	xwords = new int[len] (Line 18)	result.words = xwords (Line 25)	c-use	Definition at line 18 used in assignment
ywords	ywords = new int[len] (Line 19)	y.getAbsolute(ywords) (Line 21)	c-use	Definition at line 19 used in method call
ywords	ywords = new int[len] (Line 19)	MPN.gcd(xwords, ywords, len) (Line 22)	c-use	Definition at line 19 used in GCD computation
result	result = new BigInteger(0) (Line 23)	result.ival = len (Line 24)	c-use	Definition at line 23 used in assignment
result	result = new BigInteger(0) (Line 23)	result.words = xwords (Line 25)	c-use	Definition at line 23 used in assignment
result	result = new BigInteger(0) (Line 23)	return result.canonicalize() (Line 26)	c-use	Definition at line 23 used in return

B)

- Test Cases for All-Defs Coverage for **gcd(BigInteger y)**

Test Case	input (this, y)	Definitions Covered	Expected Output
1	(0, 5)	xval (Line 1), yval (Line 2)	5
2	(-12, -18)	xval (Line 1, Line 8), yval (Line 2, Line 10)	6
3	(7, 13)	xval (Line 12), yval (Line 16), len (Line 17, Line 22), xwords (Line 18), ywords (Line 19), result (Line 23)	1

- Test Cases for All-Defs Coverage for compareTo(BigInteger val)

Test Case	Input (this, val)	Definitions Covered	Expected Output
1	(5, 10)	None (early return)	-1
2	(-5, 10)	x_negative (Line 3), y_negative (Line 4)	-1
3	(2^100, 2^50)	x_len (Line 7), y_len (Line 8)	1

C) Data Flow Testing - Design Test Cases for All-Uses Coverage

- Test Cases for All-Uses Coverage for gcd(BigInteger y)

Test Case	Input (this, y)	du-Pairs Covered	Expected Output
1	(0, 5)	xval (Line 1 → Line 4)	5
2	(12, -18)	xval (Line 1 → Line 6), xval (Line 1 → Line 7), yval (Line 2 → Line 6), yval (Line 2 → Line 9), yval (Line 10 → Line 11)	6
3	(-12, 18)	xval (Line 8 → Line 11)	6

4	(7, 0)	xval (Line 12 → Line 17), yval (Line 2 → Line 14)	7
5	(7, 13)	yval (Line 16 → Line 17), len (Line 17 → Line 18), len (Line 17 → Line 19), len (Line 17 → Line 22), len (Line 22 → Line 24), xwords (Line 18 → Line 20), xwords (Line 18 → Line 22), xwords (Line 18 → Line 25), ywords (Line 19 → Line 21), ywords (Line 19 → Line 22), result (Line 23 → Line 24), result (Line 23 → Line 25), result (Line 23 → Line 26)	1

- Test Cases for All-Uses Coverage for compareTo(BigInteger val)

Test Cases	Input (this, val)	du-Pairs Covered	Expected Output
1	(5, 10)	None (early return)	-1
2	(-5, 10)	x_negative (Line 3 → Line 5), x_negative (Line 3 → Line 6), y_negative (Line 4 → Line 5)	-1
3	(2^100, -2^50)	None (redundant for All-Uses)	1
4	(2^100, 2^200)	x_negative (Line 3 → Line 10), x_len (Line 7 → Line 9), x_len (Line 7 → Line 10), y_len (Line 8 → Line 9), y_len (Line 8 → Line 10)	-1
5	(2^100, 2^50)	x_len (Line 7 → Line 11)	1

D) Implement and run the test cases in JUnit.

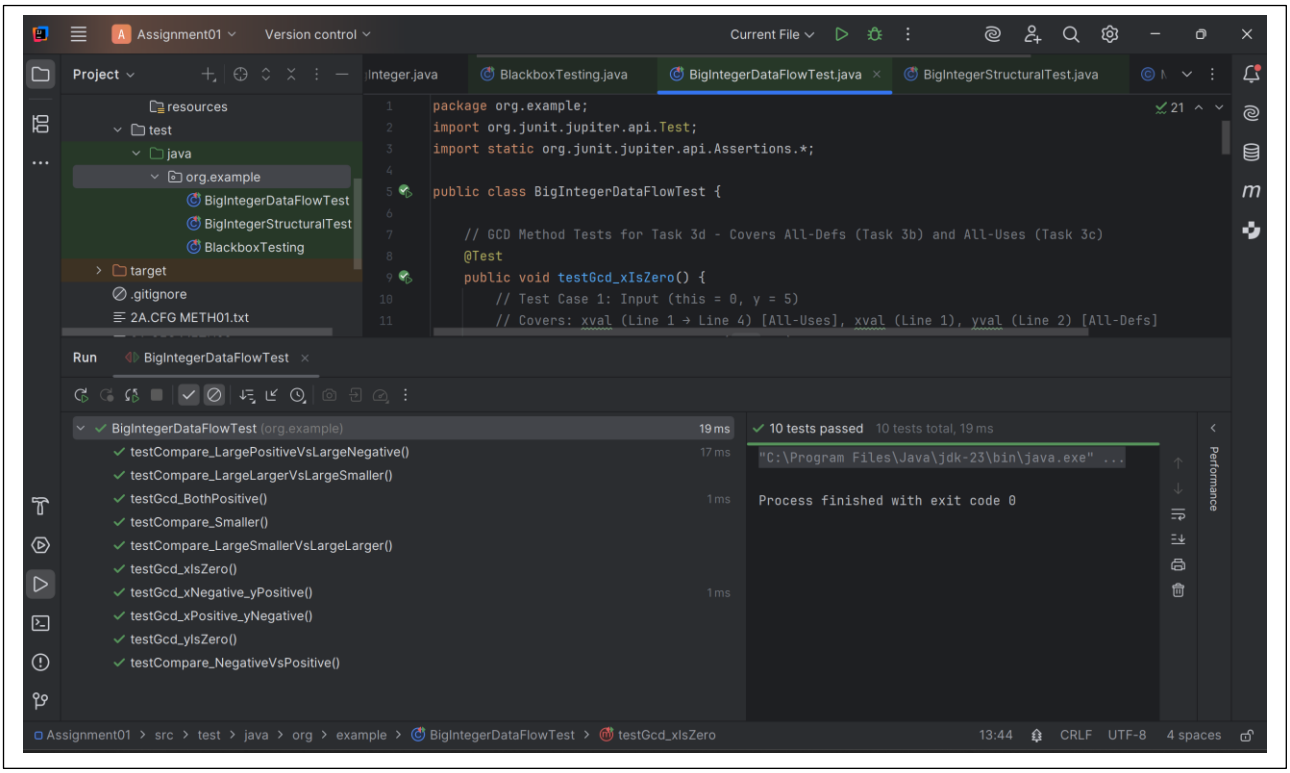


Fig: Implement of Data Flow Testing