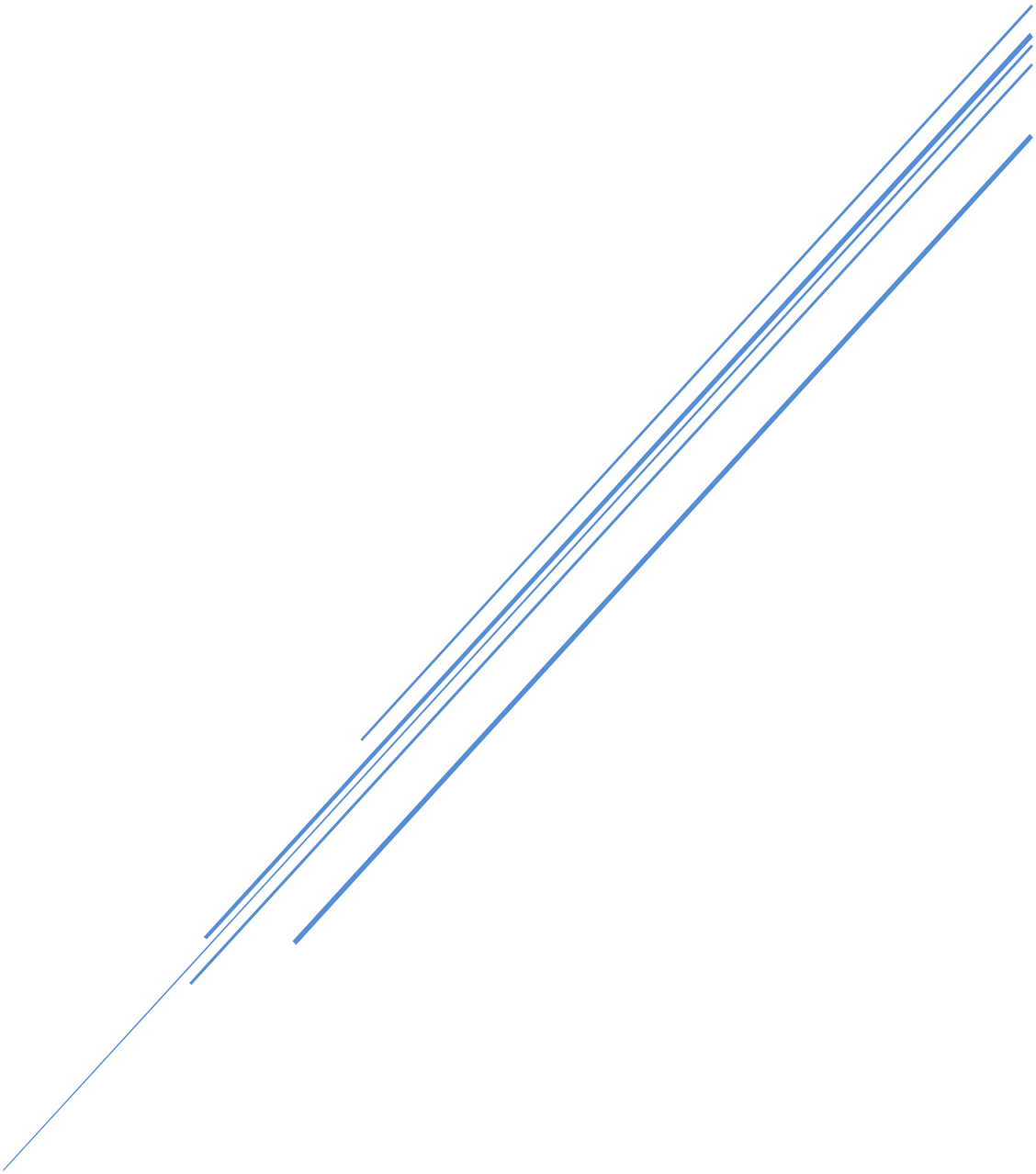
PROJECT REPORT

Assignment #1 - Group #3

C3282229, C3279166, C3305509, C3303000



University of Newcastle
SENG3320

Task 1: Black Box Testing

public BigInteger(int signum,byte[] magnitude)

Equivalence Classes:

This method has 3 equivalence classes.
INVALID: x < -1
VALID: Integers -1, 0, and 1
INVALID: x > 1

Signum can either be: -1, 0, 1

If signum = -1 or 1, then the magnitude has to contain 1 or more nonzero integers
If signum = 0, then then magnitude has to contain no nonzero integers
x < 0 = 256 - x; e.g -1 = 255

Boundary Value Analysis:

Let x = 32,767
Test Cases:

1. BigInteger(-2, x) | Should be INVALID
2. BigInteger(-1, x) | Should be **VALID**
3. BigInteger(0, x) | Should be INVALID
4. BigInteger(0, 0) | Should be **VALID**
5. BigInteger(1, x) | Should be **VALID**
6. BigInteger(2, x) | Should be INVALID

Test Case:

1. Tests the outer negative limit for signum
2. Tests the -1 limit for signum
3. Tests the 0 signum with a nonzero magnitude
4. Tests the 0 signum with a zero magnitude
5. Tests the +1 limit for signum
6. Tests the outer positive limit for signum

public BigInteger(String val, int radix)

Equivalence Classes:

Value:
Must contain variations of the first x character in the string

{0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ} where *x* = radix

For example, if radix = 4, the equivalence classes would be:
{0,1,2,3} would be an equivalence class, and
{4, …, Z} would be another equivalence class.

Overall, there's always 2 equivalence classes in String val.

Radix:
MIN_RADIX = 2, MAX_RADIX = 36;

VALID: 2 <= x <= 36,
INVALID: x > 36
INVALID: x < 2

Boundary Value Analysis:

For radix x = 16 these are the Test Cases.

1. BigInteger("123AFE",16); | Should be **VALID**
2. BigInteger("123AFZ",16); | Should be **INVALID**

Test Case:

1. Tests a valid string with the given radix, which is 16 in this case
2. Tests an invalid string with the given radix, which is 16 in this case

For radix the string "123AFEZYX" these are the Test Cases.

1. BigInteger("123AFEZYX",35); | Should be **INVALID**
2. BigInteger("123AFEZYX",36); | Should be **VALID**
3. BigInteger("123AFEZYX",37); | Should be **INVALID**

Test Case:

1. Tests the string with the lower limit of the radix with the given string
2. Tests the limit of radix with the given string
3. Tests the upper limit of radix with the given string

Public int compareTo(BigInteger val) | Usage x.compareTo(y)

Equivalence Classes:

VALID Classes
-1 where x < y
0 where x = y
+1 where x > y

INVALID CLASSES
Where X or Y are not BigInteger classes.

Boundary Value Analysis:

Test Cases:

1. **x.compareTo(y) Where:**
   X = 10, Y = 1000 | **Results in -1**
2. **x.compareTo(z) Where:**
   X = 10, Y = 10 | **Results in 0**
3. **x.compareTo(a) Where:**
   X = 10, Y = -1000 | **Results in 1**

Test Case:

1. Tests if int x is less than int y
2. Tests if int x is equal to int y
3. Tests if int x is greater than int y

Junit Outcomes:

The Junit testing resulted in all the tests outcomes being as expected, with the tests above showing the expected results.

The folder for the JUnit Tests is: Main Zip File -> JUnit -> Task 1

Task 2: White-box Testing: Structural Testing

GCD:

Describe Test Environment:

For the junit testing intellij is used for the test environment.

Test Objectives:

To have sufficient coverage for 100% statement, branch decision, condition, condition/decision and multiple condition coverage coverage.

Test Cases:

For all of the test cases the control flow diagrams showing what is covered can be found in the appendix. Each figure shows a cumulative affect e.g test 2 shows the affect of test 1. This is done so that it can be seen that each coverage is fulfilled. Due to how large some of the diagrams were we've decided to put them into a folder and submit separately. They are all included in a folder with the corresponding coverage type and test number.

Control Flow Diagram:

The control flow diagram can be found in the GCD folder with the pdf labeled "Control Flow Diagram-GCD".

Statement Coverage:

1. X=0, Y=2
2. X=-1, Y=-2
3. X=1, Y=2147483648
4. X=2147483648, Y=1
5. X= 2147483648, Y=0

Branch Decision Coverage:

1. X=0, Y=2
2. X=-1, Y=-2
3. X=1, Y=2147483648
4. X=2147483648, Y=1
5. X= 2147483648, Y=0
6. X=5, Y=5

Condition Coverage:

1. X=0, Y=2
2. X=2, Y=2
3. X=-2, Y=-2
4. X=-2147483647,Y= -2147483647
5. X=1, Y=2147483648
6. X=2147483648, Y=0
7. X=2147483648, Y=2

Condition/Decision Coverage:

1. X=0, Y=2
2. X=2, Y=2
3. X=-2, Y=-2
4. X=-2147483647,Y= -2147483647
5. X=1, Y=2147483648
6. X=2147483648, Y=0
7. X=2147483648, Y=2

Multiple Condition/Decision Coverage:

1. X=0, Y=2
2. X=2, Y=2
3. X=-2, Y=-2
4. X=-2147483647,Y= -2147483647
5. X=1, Y=2147483648
6. X=2147483648, Y=0
7. X=2147483648, Y=2
8. X=-2147483647, Y=2
9. X=-2147483647, Y=2147483648
10. X=2147483648, Y=-2147483647

Junit Outcomes:

Main Zip File -> JUnit -> Task 2 -> GCD

Each test is run once, which means there is less total tests done than the amount in total from all listed as there is some overlap when achieving 100 percent coverage. The structure of each test for GCD is simply that three big Integers are created with one being for X, Y and the expected value. Then AssertEquals is used to compare the expected value to the actual value. For our tests all of them passed.

ModInverse:

100% statement coverage

1. y = -3 x = 4
2. y = 1 x = 9
3. y = 9 x = 1
4. y = 4,234,356,765 x= 3,454,565,789
5. y = 4,234,356,765 x= -3,454,565,789
6. y = 9 x= 4
7. y= 9 x=-4

100% branch coverage

1. y = -3 x = 4
2. y = 1 x = 9
3. Y = 9 x = 1
4. 4y = 4,234,356,765
   x= 3,454,565,789
5. y = 4,234,356,765
   x= -3,454,565,789
6. y = 3,454,565,789
   x= 4,234,356,765
7. y = 9 x= 4
8. y= 9 x=-4
9. y= 9 x= 32

100% statement coverage

1. y = -4 x= 6
2. y= 9 x = 4
3. y= 1 x = 6
4. y= 9 x= 1
5. y= 9 x = -4
6. y=4 x=9
7. y= 9 x= 10
8. y= 4,234,654,889
   x= 3,234,445,654
9. y= 4,234,654,889
   x = -3,234,445,654
10. y= 9 x= 1

100% condition/decision coverage

1. y = -4 x= 6
2. y= 9 x = 4
3. y= 1 x = 6
4. y= 9 x= 1
5. y= 9 x = -4
6. y=4 x=9
7. y= 9 x= 10
8. y= 4,234,654,889 x= 3,234,445,654
9. y= 4,234,654,889 x = -3,234,445,654
10. y= 9 x= 1

100% multiple condition coverage

1. y = -4 x= 6
2. y= 9 x = 4
3. y= 1 x = 6
4. y= 9 x= 1
5. y= 9 x = -4
6. y=4 x=9
7. y= 9 x= 10
8. y= 4,234,654,889 x= 3,234,445,654
9. y= 4,234,654,889 x = -3,234,445,654
10. y= 9 x= 1

CompareTo:

Test Objectives:

To have sufficient coverage for 100% statement, branch decision, condition, condition/decision and multiple condition coverage coverage.

Test Cases:

For all of the test cases the control flow diagrams showing what is covered can be found in the appendix. Each figure shows a cumulative affect e.g test 2 shows the affect of test 1. This is done so that it can be seen that each coverage is fulfilled. Due to how large some of the diagrams were we've decided to put them into a folder and submit separately. They are all included in a folder with the corresponding coverage type and test number.

Control Flow Diagram:

See Appendix Files

Statement Coverage:

1. X= 3, Y=4
2. X=3, Y=3
3. X=5, Y=1
4. X=2147483648,Y=-2147483648
5. X=-2147483648, Y=2
6. X=2147483648, Y=2
7. X=2, Y=2147483648
8. X=2147483648, Y=2147483648

Branch decision coverage:

1. X= 3, Y=4
2. X=3, Y=3
3. X=5, Y=1
4. X=2147483648,Y=-2147483648
5. X=-2147483648, Y=2
6. X=2147483648, Y=2
7. X=2, Y=2147483648
8. X=2147483648, Y=2147483648

100% condition coverage

1. test case 1 x=3, y=4
2. test case 2 x=3, y=3
3. test case 3 x=5, y=1
4. test case 4 x=2147483648, y=-2147483648
5. test case 5 x=-2147483648, y=2
6. test case 6 x=2147483648, y=2
7. test case 7 x=2, y=2147483648
8. test case 8 x=2147483648, y=2147483648

100% condition/decision coverage

1. test case 1 x=3, y=4
2. test case 2 x=3, y=3
3. test case 3 x=5, y=1
4. test case 4 x=2147483648, y=-2147483648
5. test case 5 x=-2147483648, y=2
6. test case 6 x=2147483648, y=2
7. test case 7 x=2, y=2147483648
8. test case 8 x=2147483648, y=2147483648

Extra test case needed for 100% multiple condition coverage

x false y true: x=2, y=2147483648

Junit Outcomes:

Main Zip File -> JUnit -> Task 2 -> CompareTo

Each test is run once, which means there is less total tests done than the amount in total from all listed as there is some overlap when achieving 100 percent coverage. The structure of each test for CompareTo is simply that there are two BigIntegers created x and y for what is being compared, then an AssertEquals is used to see whether the expected answer is the same as the actual. For CompareTo all our tests passed.

Task 3: White-box Testing: Data Flow Testing

GCD

Identify all the definition-use pairs (du-pairs)

Du-pairs for variable “xval”

Du-Pair	Path(s)
(1,<2,3>)	<1,2,3>
(1,<3,5>)	<1,2,3,5>
(1,<3,6>)	<1,2,3,6>
(1,<6,7>)	<1, 2,3,6,7>
(1,7)	<1, 2,3,6,7>
(1,<6,8>)	<1, 2,3,6,8>
(1,<8,11>)	<1, 2,3,6,8,11>
(1,<8,12>)	<1, 2,3,6,8,12>
(1,14)	<1, 2,3,6,8,12,14>, <1, 2,3,6,8,11,12,14>
(1,<9,18>)	<1,2,4,9,18>
(1,18)	<1,2,4,9,18>
(11,14)	<11,12,14>
(13,14)	<13,14>
(7,9)	<7,4,9>
(7,<9,18>)	<7,4,9,18>

Du-pairs for variable “y.words”

Du-Pair	Path(s)
(1,<6,7>)	<1,2,3,6,7>
(1,<6,8>)	<1,2,3,6,8>
(1,<4,9>)	<1,2,4,9>
(1,<4,10>)	<1,2,4,10>

Du-pairs for variable “words”

Du-Pair	Path(s)
(1,<2,3>)	<1,2,3>
(1,<2,4>)	<1,2,4>

Du-pairs for variable “yval”

Du-Pair	Path(s)
(1,<2,3>)	<1,2,3>
(1,<3,5>)	<1,2,3,5>
(1,<3,6>)	<1,2,3,6>
(1,<6,7>)	<1, 2,3,6,7>
(1,<6,8>)	<1,2,3,6,8>
(1,<12,13>)	<1, 2,3,6,8,12,13>
(1,<12,14>)	<1, 2,3,6,8,12,14>
(1,13)	<1, 2,3,6,8,12,13>
(1,14)	<1, 2,3,6,8,12,14>, <1, 2,3,6,8,12,13,14>
(1,<10,15>)	<1,2,4,10,15>
(1,15)	<1,2,4,10,15>
(1,<10,16>)	<1,2,4,10,16>
(1,<9,17>)	<1,2,4,9,17>
(1,17)	<1,2,4,9,17>
(1,<9,18>)	<1,2,4,9,18>
(13,14)	<13,14>
(15,<9,17>)	<15,9,17>

Du-pairs for variable “len”

Du-Pair	Path(s)
(17,19)	<17,19>
(18,19)	<18,19>
(17,22)	<17,19,20,21,22>
(18,22)	<18,19,20,21,22>
(21,22)	<21,22>

Du-pairs for variable “result”

Du-Pair	Path(s)
(22,22)	<22>
(22,23)	<22,23>

Du-pairs for variable “xwords”

Du-Pair	Path(s)
(19,20)	<19,20>
(19,21)	<19,20,21>
(19,22)	<19,20,21,22>

Du-pairs for variable “ywords”

Du-Pair	Path(s)
(19,20)	<19,20>
(19,21)	<19,20,21>

Design test cases to achieve All-Defs coverage

Nodes that have Declarations (defs), 1,7,11,13,15,17,18,19, 21,22

Test cases:
Bold nodes are nodes that correspond with ‘defs’

BigInt X = -1, BigInt Y = -1
Coverage: **1**,2,3,6,8,**11**,12,**13**,14

BigInt x = 10, BigInt Y = -2^{31}.
Coverage: **1**,2,3,6,**7**,4,10,**15**,9,**18**,**19**,20,**21**,**22**,23

Bigint x = 0, BigInt y = 2147483650
Coverage: **1**,2,4,10,**15**,9,**17**,**19**,20,**21**,**22**,23

These three (3) tests cover variable definitions throughout the GCD Function

Design test cases to achieve All-Use coverage

Nodes that have Variable Uses: 2,3,4,5,6,8,9,10,12,14,16,20,23

Test cases:
Bold nodes are nodes that correspond with variable uses

BigInt x = 10, BigInt Y = -2^{31}.
Coverage: 1,**2,3,6**,7,**4,10**,15,**9**,18,19,**20**,21,22,**23**

BigInt X = -1, BigInt Y = -1
Coverage: 1,**2,3,6,8**,11,**12**,13,**14**

BigInt X =0, BigInt Y = 0
Coverage: 1,**2,3,5**

BigInt X =2147483650, BigInt Y = 0
Coverage: 1,**2,4,10,16**

These Four (4) tests cover variable definitions throughout the GCD Function

CompareTo

Identify all the definition-use pairs (du-pairs)

Du-pairs for variable “x”

Du-Pair	Path(s)
(1,3)	<1,2,3>
(1,<3,4>)	<1,2,3,4>
(1,<3,5>)	<1,2,3,5>
(1,5)	<1,2,3,5>
(1,<5,6>)	<1,2,3,5,6>
(1,<5,7>)	<1,2,3,5,7>
(1,8)	<1,2,8>
(1,10)	<1,2,8,9,10>
(1,14)	<1,2,8,9,10,14>
(1,20)	<1,2,8,9,10,14,16,17,19,20>

Du-pairs for variable “y”

Du-Pair	Path(s)
(1,3)	<1,2,3>
(1,<3,4>)	<1,2,3,4>
(1,<3,5>)	<1,2,3,5>
(1,5)	<1,2,3,5>
(1,<5,6>)	<1,2,3,5,6>
(1,<5,7>)	<1,2,3,5,7>
(1,8)	<1,2,8>
(1,16)	<1,2,8,9,10,14,16>
(1,<16,17>)	<1,2,8,9,10,14,16,17>
(1,<16,18>)	<1,2,8,9,10,14,16,18>
(1,17)	<1,2,8,9,10,14,16,17>
(1,20)	<1,2,8,9,10,14,16,17,19,20>

Du-pairs for variable “x_negative”

Du-Pair	Path(s)
(8,9)	<8,9>
(8,<9,10>)	<8,9,10>
(8,<9,11>)	<8,9,11>
(8,11)	<8,9,11>
(8,12)	<8,9,11,12>
(8,<11,13>)	<8,9,11,13>
(8,<21,22>	<8,9,10,14,16,17,19,21,22> <8,9,10,14,16,18,19,21,22> <8,9,10,15,16,17,19,21,22> <8,9,10,15,16,18,19,21,22>
(8,<21,23>	<8,9,10,14,16,17,19,21,23> <8,9,10,14,16,18,19,21,23> <8,9,10,15,16,17,19,21,23> <8,9,10,15,16,18,19,21,23>

Du-pairs for variable “y_len”

Du-Pair	Path(s)
(17,19)	<17,19>
(18,19)	<18,19>
(17,<19,21>)	<17,19,21>
(18,<19,21>)	<18,19,21>
(17,21)	<17,19,21>
(18,21)	<18,19,21>
(17,<21,22>)	<17,19,21,22>
(18,<21,22>)	<18,19,21,22>
(17,<21,23>)	<17,19,21,23>
(18,<21,23>)	<18,19,21,23>

Du-pairs for variable “y_negative”

Du-Pair	Path(s)
(8,9)	<8,9>
(8,<9,10>)	<8,9,10>
(8,<9,11>)	<8,9,11>

Du-pairs for variable “x_len”

Du-Pair	Path(s)
(14,19)	<14,16,17,19> <14,16,18,19>
(14,<19,20>)	<14,16,17,19,20> <14,16,18,19,20>
(14,20)	<14,16,17,19,20> <14,16,18,19,20>
(14,<19,21>	<14,16,17,19,21> <14,16,18,19,21>
(14,21)	<14,16,17,19,21> <14,16,18,19,21>
(14,<21,22>)	<14,16,17,19,21,22> <14,16,18,19,21,22>
(14,<21,23>	<14,16,17,19,21,23> <14,16,17,18,21,23>
(15,19)	<15,16,17,19> <15,16,18,19>
(15,<19,20>)	<15,16,17,19,20> <15,16,18,19,20>
(15,<19,21>	<15,16,17,19,21> <15,16,18,19,21>
(15,21)	<15,16,17,19,21> <15,16,18,19,21>
(15,<21,22>)	<15,16,17,19,21,22> <15,16,18,19,21,22>
(15,<21,23>	<15,16,17,19,21,23> <15,16,18,19,21,23>

Design test cases to achieve All-Defs coverage

Nodes that have Declarations (defs), 1,8, 14, 15, 17, 18
Test cases: Bold nodes are nodes that correspond with variable defs
BigInt x = 10, BigInt Y = 2^{31}+10.
Coverage: **1**,2,**8**,9,10,**15**,16,**17**,19,21,22
Expected -1

BigInt x = 2^{31}+10, BigInt Y = 10.
Coverage: **1**,2,**8**,9,10,**14**,16,**18**,19,21,23
Expected 1

These Two (2) tests cover variable definitions throughout the CompareTo

Design test cases to achieve All-Use coverage

Nodes that have Declarations (defs), 2,3,4,5,6,7,9,10,11,12,13,16,19,20,21,22,23

Test cases: Bold nodes are nodes that correspond with variable Uses

BigInt x = 2^{31}+10, BigInt Y = 10.
Coverage: 1,**2**,8,**9,10**,14,**16**,18,**19,21,23**
Expected 1

BigInt x = 1, BigInt Y = 1.
Coverage: 1,**2,3,6**
Expected 0

BigInt x = -1, BigInt Y = 1.
Coverage: 1,**2**,**4**
Expected -1

BigInt x = 1, BigInt Y = -1.
Coverage: 1,**2**,**3,5,7**
Expected 1

BigInt x = -2^{31}+10, BigInt Y = 10.
Coverage: 1,**2**,8,**9,11,12**
Expected -1

BigInt x = 2^{31}+10, BigInt Y = -10.
Coverage: 1,**2**,8,**9,11,13**
Expected 1

BigInt x = 2^{31}+10, BigInt Y = 2^{31}+10.
Coverage: 1,**2**,8,**9,10,**14**,16,**17**,19,20**
Expected 0

BigInt x = 2^{31}+10, BigInt Y = 2^{31}+100.
Coverage: 1,**2**,8,**9,10,**14**,16,**17**,19,21,22**
Expected -1

BigInt x = 2^{31}+100, BigInt Y = 2^{31}+10.
Coverage: 1,**2**,8,**9,10,**14**,16,**17**,19,21,23**
Expected 1

These Nine (9) tests cover variable uses throughout the CompareTo

Junit Outcomes:

The Junit testing resulted in all the tests passing.
Main Zip File -> JUnit -> Task 3