THE UNIVERSITY OF NEWCASTLE AUSTRALIA

SCHOOL OF INFORMATION AND PHYSICAL SCIENCES

School of Information and Physical Sciences

SENG3320/6320 Software Verification and Validation

Assignment 1

This assignment is to be completed in groups.

Due on ***Friday, 2 May 2025, 11:59pm****, electronically via the "Assignment 1" submission link in Canvas.*

Total 100 marks
(Weight 25% of the overall course assessment)

Aims

This assignment aims to enhance students' practical understanding of software testing by applying both black-box and white-box techniques to real-world Java code. Through tasks involving structural and data flow testing, students will learn to design effective test cases, achieve various code coverage criteria, and execute automated tests using JUnit. The assignment also emphasizes the importance of clear documentation and reporting of test results. By working collaboratively in groups, students will further develop their skills in systematic software verification and validation within a team-based environment.

Project Description

The BigInteger class is a Java math class used for performing mathematical operations involving very large integers that exceed the limits of all primitive data types. For example, two large integers such as 5454564684456454684646454545 and 4256456484464684864864864864 can be represented and manipulated using BigInteger, and their sum would be 9711021168921139549511319409.

You can find the API specification at:
https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html

The source code is available at:

https://developer.classpath.org/doc/java/math/BigInteger-source.html

(Note: This implementation is from the GNU Classpath 0.95 framework. The required files—BigInteger.java and MPN.java—are provided on Canvas.)

In this assignment, you will perform both black-box and white-box testing on selected methods from the BigInteger class.

1. (24 Marks) Blackbox Testing

Consider the following three methods from the BigInteger class (API specifications provided in the Appendix):

```
public BigInteger(int signum, byte[] magnitude)
public BigInteger(String val, int radix)
public int compareTo(BigInteger val)
```

For each method:

a. (**15 Marks**) Design test cases using the Equivalence Partitioning technique.
- Clearly state the equivalence classes.
- For each test case, specify the partition being tested, the corresponding test inputs, and the expected outputs.

b. (**9 Marks**) Implement and run the test cases in JUnit.

2. (42 Marks) Whitebox Testing: Structural Testing

Consider the following two methods from the BigInteger class (source code in Appendix):

```
public BigInteger gcd(BigInteger y)
private static int compareTo(BigInteger x, BigInteger y)
```

(Note: compareTo is a private method. You can test it through the public method public int compareTo(BigInteger val), which calls it internally.)

For each method:

a. (**6 Marks**) Draw a control flow graph.
b. (**4 Marks**) Design test cases to achieve 100% statement coverage.
c. (**4 Marks**) Design test cases to achieve 100% branch (decision) coverage.
d. (**4 Marks**) Design test cases to achieve 100% condition coverage.
e. (**4 Marks**) Design test cases to achieve 100% condition/decision coverage.
f. (**6 Marks**) Design test cases to achieve 100% multiple condition coverage.
g. (**6 Marks**) Design test cases to achieve 100% modified condition/decision coverage.
h. (**8 Marks**) Implement and run the test cases in JUnit.

Note: If 100% coverage is not achievable, report the actual coverage achieved and explain why 100% could not be attained. Also, list any reasonable assumptions made during the test design.

3. (26 Marks) Whitebox Testing: Data Flow Testing

Consider the same methods as in Task 2:

```
public BigInteger gcd(BigInteger y)
private static int compareTo(BigInteger x, BigInteger y)
```

(Again, the private method should be tested through the corresponding public method.)

For each method:

a. (**6 Marks**) Identify all definition-use (du) pairs.
b. (**6 Marks**) Design test cases to achieve All-Defs coverage.
c. (**6 Marks**) Design test cases to achieve All-Uses coverage.
d. (**8 Marks**) Implement and run the test cases in JUnit.

4. **(8 Marks) Report writing and presentation.**
- The report should be well structured.
- Provide clear explanations of the results where applicable.
- Include screenshots of the JUnit execution results in the report.

Submission

All assignments must be submitted via Canvas. If multiple submissions are made, only the latest one will be graded. This assignment consists of two parts: (1) **the project report and**

source code, and (2) **the peer evaluation form (not graded)**. The project report and source code should be submitted by only one group member on behalf of the entire group (in a single **.zip file**). However, each group member must individually submit a peer evaluation form that reflects the contributions of all team members. The peer evaluation form must be submitted through the separate “Assignment 1 Peer Evaluation” submission link on Canvas.

Your submission (.zip file) must include the following:

(1) An assignment cover sheet.
(2) A PDF testing report that contains answers to all questions (excluding the source code of the JUnit testing files).
(3) A folder containing the JUnit test project, including: a). The complete project folder structure; b). A README file specifying: Instructions for compiling and executing the tests; Any external libraries (and versions) or dependencies used; and The Java version and IDE (if any) used for development.

The mark for an assessment item submitted after the designated time on the due date, without an approved extension of time, will be reduced by 10% of the possible maximum mark for that assessment item for each day or part day that the assessment item is late. **Note: this applies equally to week and weekend days.**

Use of Generative AI tools

The use of generative AI tools is permitted for this assessment under the following conditions:

1) Generative AI tools may be used to assist with understanding concepts; however, they must not be used to directly generate answers to assignment questions. Directly copying and pasting AI-generated content into your report is considered plagiarism.

2) If generative AI tools are used (excluding cases where they are used solely for language assistance), you must document the following in a separate section at the end of your report:

- The name of the tool(s) used
- The prompts or queries submitted
- The corresponding AI-generated output (as copy & paste or screenshots)

3) Be mindful of the limitations of generative AI tools, such as hallucination (i.e., generating incorrect or misleading information). Always critically evaluate the output.

Plagiarism

All work must be your own. Plagiarism, including copying from other groups or unauthorized sources, will be handled in accordance with the university’s academic integrity policy.

Appendix

API Specifications:

public BigInteger(int signum,byte[] magnitude)

Translates the sign-magnitude representation of a BigInteger into a BigInteger. The sign is represented as an integer signum value: -1 for negative, 0 for zero, or 1 for positive. The magnitude is a byte array in *bigendian* byte-order: the most significant byte is in the zeroth element. A zero-length magnitude array is permissible, and will result in a BigInteger value of 0, whether signum is -1, 0 or 1.

Parameters:

`signum` - signum of the number (-1 for negative, 0 for zero, 1 for positive).

`magnitude` - big-endian binary representation of the magnitude of the number.

Throws:

`NumberFormatException` - `signum` is not one of the three legal values (-1, 0, and 1), or `signum` is 0 and `magnitude` contains one or more non-zero bytes.

public BigInteger(String val, int radix)

Translates the String representation of a BigInteger in the specified radix into a BigInteger. The String representation consists of an optional minus or plus sign followed by a sequence of one or more digits in the specified radix. The character-to-digit mapping is provided by `Character.digit`. The String may not contain any extraneous characters (whitespace, for example).

Parameters:

`val` - String representation of BigInteger. `radix`

- radix to be used in interpreting `val`.

Throws:

`NumberFormatException` - `val` is not a valid representation of a BigInteger in the specified radix, or `radix` is outside the range from `Character.MIN_RADIX` to `Character.MAX_RADIX`, inclusive.

public int compareTo(BigInteger val)

Compares this BigInteger with the specified BigInteger. This method is provided in preference to individual methods for each of the six boolean comparison operators (<, ==, >, >=, !=, <=). The suggested idiom for performing these comparisons is: `(x.compareTo(y)` <*op*> `0)`, where <*op*> is one of the six comparison operators.

Specified by:

`compareTo` in interface `Comparable<BigInteger>` **Parameters:**

`val` - BigInteger to which this BigInteger is to be compared.

Returns:

-1, 0 or 1 as this BigInteger is numerically less than, equal to, or greater than `val`.

Source Code:

public BigInteger gcd(BigInteger y)

```
public BigInteger gcd(BigInteger y)
{
  int xval = ival;
  int yval = y.ival;
  if (words == null)
    {
  if (xval == 0)
    return abs(y);
  if (y.words == null
      && xval != Integer.MIN_VALUE && yval != Integer.MIN_VALUE)
    {
      if (xval < 0)
        xval = -xval;
      if (yval < 0)
        yval = -yval;
      return valueOf(gcd(xval, yval));
    }
  xval = 1;
    }
  if (y.words == null)
    {
  if (yval == 0)
    return abs(this);
  yval = 1;
    }
  int len = (xval > yval ? xval : yval) + 1;
  int[] xwords = new int[len];
  int[] ywords = new int[len];
  getAbsolute(xwords);
  y.getAbsolute(ywords);
  len = MPN.gcd(xwords, ywords, len);
  BigInteger result = new BigInteger(0);
  result.ival = len;
  result.words = xwords;
  return result.canonicalize();
}
```

private static int compareTo(BigInteger x,BigInteger y)

```
private static int compareTo(BigInteger x, BigInteger y)
{
  if (x.words == null && y.words == null)
    return x.ival < y.ival ? -1 : x.ival > y.ival ? 1 : 0;
  boolean x_negative = x.isNegative();
  boolean y_negative = y.isNegative();
  if (x_negative != y_negative)
    return x_negative ? -1 : 1;
  int x_len = x.words == null ? 1 : x.ival;
  int y_len = y.words == null ? 1 : y.ival;
  if (x_len != y_len)
    return (x_len > y_len) != x_negative ? 1 : -1;
  return MPN.cmp(x.words, y.words, x_len);
}
```