Cosc 1P03 Assignment 3
Abstract Data Types

(Due Date March 10th 9:00 a.m. No Lates accepted)

Objective

Develop an abstract data type and test harness.

Background

Consider the case where large integer numbers are required for calculations which exceed the capacity of resident types such as long within java. These numbers can easily exceed 100's of digits. Number objects such as these can be compared, printed, and have simple arithmetic operations applied.

The Assignment

Develop an abstract data type which will be part of the package BigNumbers. This implementation should implement the below interface.

```
package BigNumbers;

public interface BNum {

      // Create a clone of this.
      public BNum clone();

      /** Returns true if 'this' = n*/
      public boolean equals ( BNum n);

      /** Returns true if 'this' < n */
      public boolean lessThan (BNum n);

      /** returns 'this' + n */
      public BNum add (BNum n);

      /** returns 'this' - n */
      public BNum sub (BNum n);

      /** Returns the sign of this BigNumber object */
      public int getSign();

      //Returns the digit i of this object, digit 0 is LSD.
            public int getDigit(int i);

}
```

You will develop two implementations, the first ConBNum the second LinkBNum. These will use arrays and linked list implementations respectfully. Each implementation should support toString.

Constructors for each implementation should support the default constructor, a constructor which accepts a **long**, and one that accepts a **String**. The default constructor creates an object with a +0 value. Use constructor chaining where appropriate.

Two possible exceptions can be thrown. BadNumberFormatException shall be thrown in the case where the constructor which accepts a **String** input finds illegal characters and cannot successfully parse the input to a valid BNum. E.g. **`new ConBNum("1234t80");`** would result in an obvious error. The second constructor DigitOutOfRangeException shall be thrown if **`getDigit(int i)`** is called and **i** is out of range for this object. For both constructors, write these so that they support the inclusion of a message.

Once complete all classes should be part of the same package. Be sure to use appropriate modifiers on the classes and instance variables.

Test Harness

You will be required to write your own test harness to ensure that your implementations are correct. The test harness should exercise all methods, constructors and exceptions of each implementation. In addition, show that you can do mixed operations on BNum objects of both implementations. E.g.

```
Bnum      A,B,C;
A=new ConBNum("1234");
B=new LinkBNum(1234);
C=A.add(B);
```

The test harness should be part of the package TestPackage. Thus you will need to import BigNumbers into the test harness for use. It will be the students responsibility to fully test their implementation and prove that it works as specified.

Marking Scheme

Marks will be awarded for following instructions as well as good coding practices. Here is a list but not exclusive, of where marks can be awarded.

1. Implementations using the correct underlying data structure.
2. Correct modifiers on both classes and instance variables.
3. Exceptions which pass personalized diagnostic messages
4. Test harness which tests all aspects of all implementations of the ADT
5. Following all the above instructions.

Submission

Submit copies of all code written, the ADT and Test Harness. Include a copy of your output.