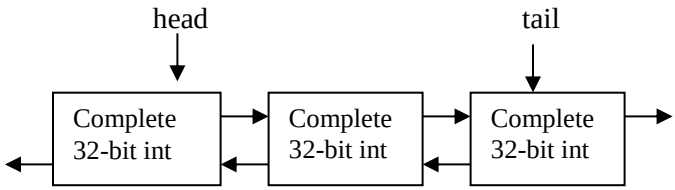
CSC205AB
Program3: InfiniteInt

Background:

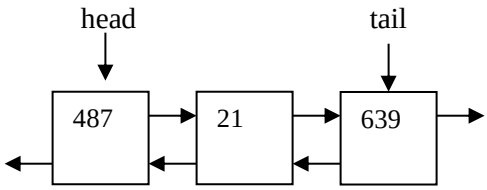
Please use the doubly linked list you developed (DLList.java). Let me know if you do not have one and I will provide one for you. As you conceptualize the program, you will see why a doubly linked list is used.

The Program:

Write a new class called InfiniteInt. We can (theoretically) store an "infinite" integer by linking together nodes that hold actual complete integers. So it will look like this:



For this program, just store 3 digits in each node (concept is the same, but we won't have to generate hundreds of digits to test it). For example, the integer 487,021,639 will be stored like this:



As you can see, it uses a doubly linked list; therefore, make your InfiniteInt class a subclass of a DLList that holds Integers. Be sure that all of DLList's data (head and tail) can be inherited from the superclass. Since it also will have to have a compareTo (see requirements below), your InfiniteInt should be defined as:

public class InfiniteInt extends DLList<Integer> implements Comparable<InfiniteInt>

Since your InfiniteInt is a subclass of DLList, it will inherit DLList's methods, but you will also have to implement the following methods/constructors:

- A default constructor that takes no arguments and just builds a "null" linked list (it could call its super() which just sets head and tail to null).

- A constructor that receives a String as an argument and builds the linked list. If an InfiniteInt is created as follows:

  InfiniteInt myList = new InfiniteInt("487,021,639");

  then the list should be built as shown above.

  This constructor should also check for illegal Strings (containing a non-digit). If encountered, throw an IllegalArgumentException(<message of your choice>).

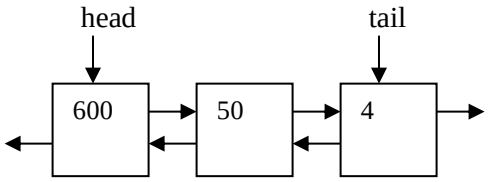
  Note that you can build the list by using the methods already available from the superclass.

- An equals method that receives another Object and returns true if it is equal to this InfiniteInt. It is considered equal if the numbers themselves are exactly the same. You will have to start with the same exact logic that we have used in previous implementations of .equals. The actual part to determine equality will be very easy (think about it and what you have to work with).

- A toString() method that will override the one in the superclass. If the InfiniteInt is empty, your toString() method should

  throw new IllegalStateException(<whatever meaningful String you want>);

  Otherwise, it returns the String representation of the InfiniteInt with commas and leading 0's inserted correctly. You cannot count on the original String representation being available (the InfiniteInt could have been generated using the add method below) so you have to go through the nodes and determine what to print. Note that the following InfiniteInt's toString() should return "600,050,004".



- A method called isEven() which returns true if the InfiniteInt is an even number. First check to see if it is empty; if so, the isEven() method should:

  throw new IllegalStateException(<whatever meaningful String you want>);

- A method called isDivisibleBy10() which returns true if the InfiniteInt is evenly divisible by 10. First check to see if it is empty; if so, the isDivisibleBy10() method should:

  throw new IllegalStateException(<whatever meaningful String you want>);

- A method called isDivisibleBy1000() which returns true if the InfiniteInt is evenly divisible by 1000. First check to see if it is empty; if so, the isEven() method should:

  throw new IllegalStateException(<whatever meaningful String you want>);

- A compareTo(InfiniteInt o) that will implement the Comparable interface (please actually put "implements Comparable<InfiniteInt> in the class definition). So compareTo will return a positive (Java) int if the InfiniteInt is greater than what is passed in, a negative (Java) int if the InfiniteInt is less than what is passed in, and 0 if the InfiniteInt is the same as what is passed in. This code should work in a driver program:

```
InfiniteInt int1 = new InfiniteInt("24");
InfiniteInt int2 = new InfiniteInt("6");
InfiniteInt int3 = new InfiniteInt("24");
System.out.println(int1.compareTo(int2));    //should print a positive (Java) int
System.out.println(int2.compareTo(int1));    //should print a negative (Java) int
System.out.println(int1.compareTo(int1));    //should print 0
System.out.println(int1.compareTo(int3));    //should print 0
```

- A static add method which will receive 2 InfiniteInts as arguments, add them up, and return a new InfiniteInt with the total in it. If you think about this and try a few examples on paper, you will see how to do it – you must traverse each number backwards (using the prev link in the doubly linked list) and add each digit, carrying to the next place when necessary. Make sure that you carry correctly and handle the case when one list is longer than the other. This code should work in a driver program:

```
InfiniteInt int1 = new InfiniteInt("646,746,734");
InfiniteInt int2 = new InfiniteInt("543,534");
InfiniteInt int3;
int3 = InfiniteInt.add(int1, int2);
System.out.println(int3);                                    //should print 647,290,268
```

Hints:

- Until you get your toString() to work, it can just call super.toString() to call DLList's toString() and actually see what is in the nodes.
- There are a number of ways to "parse" a String like "43,453,126". One way is to use an instance of StringTokenizer with "," defined as the delimiter. See the StringTokenizer demo.

Comments and formatting: Please use the Java conventions for variable names, indenting, and formatting. Each class should have an opening comment which briefly describes the class and includes your name and class on a separate line. Each method should have a short opening comment which describes it. "Sections" of code or parts that are tricky should have comments

Extra Credit (optional): Also write a method for the InfiniteInt class called isDivisibleBy6() which returns true if the InfiniteInt is evenly divisible by 6. In Math, a number is evenly divisible by 6 if it is evenly divisible by 2 and evenly divisible by 3. A number is evenly divisible by 3 if the sum of its individual digits is evenly divisible by 3.

To get credit for this, it has to work in the context of InfiniteInts. Since an InfiniteInt can represent a giant integer (500 nodes each containing 999 or more for example), you cannot simply use a variable to store the sum of the digits since that variable might overflow. This is something I cannot test for on Hypergrade – I tried creating a new InfiniteInt and then doing the following:

```
for (int i=0; i< 341606371735362067; i++) //number of nodes it would take to make the counter overflow
        testerInfiniteInt.addLast(999);
```

but it clearly took too long to run. So I will look at your solution and evaluate it as to whether it would run on a test Infinite with that many nodes…

Please submit on HyperGrade: your InfiniteInt.java file and the DLList.java file that you use.