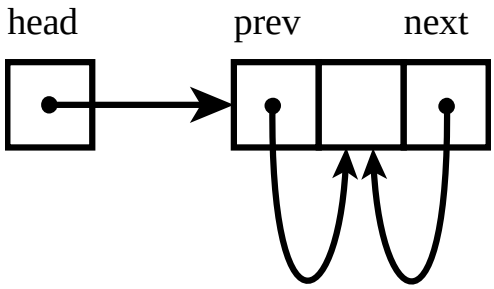
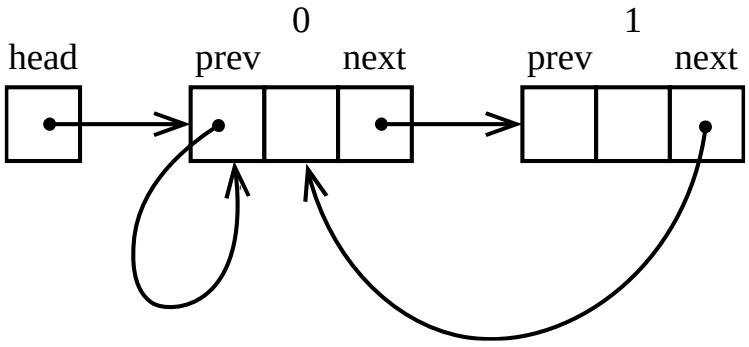
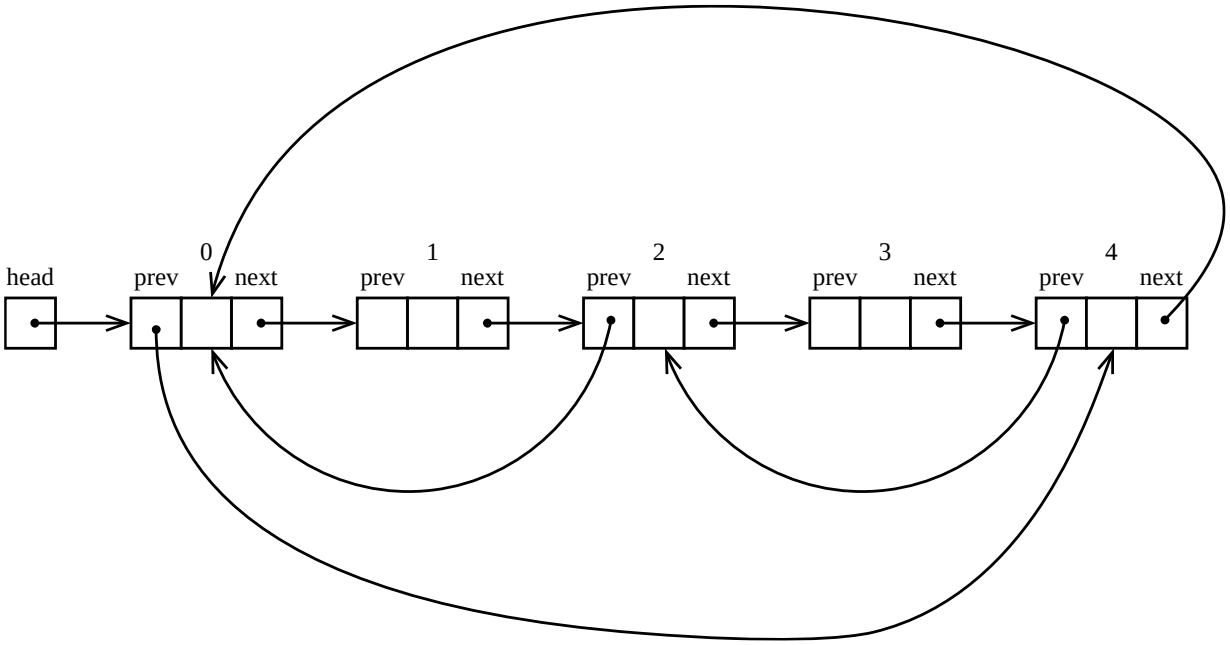
(50 points) An "Amusing" Collection Implementation

In this first part of the assignment you will be required to write a class that implements the java.util.List interface using a special "amusing" version of the linked list. Your implementing class must be called AmusingLinkedList. (Note that after you finish this assignment, you may feel like changing the word "amusing", but this assignment really will help you learn linked lists. In this implementation, from the client's point-of-view, the first item is found at index 0, the second at index 1 and the last is at an index equal to the size of the list minus 1. (This is the same as the java.util.List specification.)

Your implementation must use a linked-list to store the data in a particular way. Forward links in the list link to the next node in the list as in the standard singly-linked list. The difference is that the last node in the list is linked to the first node, to form a circular linked list, and previous links for each node are only valid for nodes at even indexes of the list. The previous fields of odd index nodes are null, and the previous field of even indexed nodes are linked with the previous even indexed node. (Remember, indexes in this list start at 0 and that 0 is considered even for this assignment, **and the prev field of index 0 references the last even node in the list.**) Several sample lists are shown below. Pay particular attention to the degenerate lists where there are few nodes. **An empty list is represented by a null head reference**.







Complete documentation for Java List interface API can be found on https://docs.oracle.com/javase/8/docs/api/java/util/List.html website and many other places. You are expected to read this documentation to determine the behavior of the methods. In the information below, pay close attention to the requirements for efficiency when stated. When you complete your implementation of this interface, the results from your tests should match identically to those of any of the implementations of the List interface. **Also note that for this part of the assignment, your List implementation is not allowed to subclass any other classes and your Node inner class must be public, named Node and may only have 3 fields named next, prev, and data. The Node class must also contain three get methods named getData, getNext, and getPrev that return the values in the fields data, next, and prev, respectively. To reiterate, the Node class must also be a public inner class of the AmusingLinkedList class. Using any member variables or local variables that reference implementations of the List interface or other interfaces or classes that implement List does not satisfy this assignment and such submissions shall not receive any credit.**

You will also need to create a class called AmusingListIterator that implements the ListIterator<E> interface. The AmusingListIterator class must be implemented as an inner class of the AmusingLinkedList class. **You are not required to implement the optional methods in the ListIterator and Iterator interfaces**, but it may be helpful for the client that you write in the second part of this assignment.

Also be sure to look carefully at the toString method for the AmusingList class and implement it as described. Be sure to understand what happens when you add and delete nodes from this list. Make sure when you design your add and delete methods that the links are correct after any possible add or delete operations. Also beware that the "previous" method in the Collection interface still returns the previous item in the list, not the previous item that is in the prev link of the node.

You are also required to write and include one additional method in the AmusingLinkList class

that given an index into the list, you return the actual Node object at that index. This method is required for grading, and most tests will fail if you do not include this method in your class. The method signature is:

public Node getNodeAtIndex(int index);

No method in your implementation of the List interface may have greater than O(n) run time, and the methods "boolean add(E o)", "int size()", "boolean isEmpty()", and "clear()" must run in O(1) (constant) time. In addition to the methods required by the List Interface, you must also override the toString method. The definition for this method is given below.

public String toString()

Returns a string representation of the class. The toString method for this class must return the following data, starting with the head and traversing forward through the list.

- List index of the current node in the iteration
- List index of the node that the previous link references. (Print -1 if the previous reference is null)
- List index of the node referenced by the next link. (Print -1 if the next reference is null).
- the String returned when toString is called on the data object for the node, or "null" if the data item is null.

The following is an example of the information that is printed for a List of 6 Entries, "Jim", "Mary", "Jennifer", null, "Denise", "Phil"

0 4 1 Jim

1 -1 2 Mary

2 0 3 Jennifer

3 -1 4 null

4 2 5 Denise

5 -1 0 Phil

(50 points) An "Amusing" Precise Number Class

Using a class that implements the List interface, write a class called AmusingPreciseNumber that is used to represent very large or very small numbers with arbitrary precision. **This class may only use the List objects in a HAS-A relationship**. This means an AmusingPreciseNumber contains a List (perhaps more than one) as a member variable, but does not implement the List interface or any of its methods. Since this is an interface, you may use either a List implementation from the standard Java libarary, or the List implementation you created as defined above. The really cool student will use their own List implementation from above. However, if for some reason you cannot complete that part, you may use a Java 8 List implementation for this part.

Pay particular attention to the method and constructor signatures described below. You must implement these methods exactly as defined here, or it will be impossible for us to test your implementation.

Constructors:

public AmusingPreciseNumber(int numb)

Create an AmusingPreciseNumber from an int type.

public AmusingPreciseNumber(String numb)
Create an AmusingPreciseNumber from a String. The formatting of the string is some number of digits with an optional decimal point. Your constructor is required to throw a runtime exception if the string does not have a valid syntax. Valid strings do include 0, 0.0, 0000, 00000123, 00000123.000001000, -23432, +1234., and +1234555. That is, leading or trailing zeros, a single leading plus or minus sign, and no plus or minus sign are all valid numbers. In effect, any reasonable string of digits (no matter how long) that can be interpreted as a number is valid.

public AmusingPreciseNumber(Reader r)
The same as the string constructor except the input comes from arbitrary Reader. This means that there is no bound on the number of digits for this constructor. The format is similar to the String constructor except that a whitespace character is treated as a termination of the input and no further data is read from the stream once this whitespace character is read. Leading whitespace characters are ignored. We will test that this constructor can handle at least 100,000 digits of precision.

public AmusingPreciseNumber(AmusingPreciseNumber numb)
A simple copy constructor. It is required that this be a deep copy. We will test this.

Methods:

public void add(AmusingPreciseNumber numb)
Add numb to this AmusingPreciseNumber

public void subtract(AmusingPreciseNumber numb)
Subtract numb from this AmusingPreciseNumber

public void negate()
Negate this AmusingPreciseNumber

public void abs()
Compute and store the absolute value of this AmusingPreciseNumber

The following are static methods for this class.

public static AmusingPreciseNumber add(
AmusingPreciseNumber numb1,
AmusingPreciseNumber numb2)
Return an AmusingPreciseNumber that is the sum of numb1 and numb2. Numb1 and numb2 are unchanged.

public static AmusingPreciseNumber subtract(
AmusingPreciseNumber numb1
AmusingPreciseNumber numb2
Return an AmusingPreciseNumber that is the difference of numb1 and numb2 (numb1 minus numb2)

Numb1 and numb2 are unchanged

public static AmusingPreciseNumber negate(
AmusingPreciseNumber numb)
Return an AmusingPreciseNumber that is the negative of numb and leave numb unchanged.

public static AmusingPreciseNumber abs(
AmusingPreciseNumber numb)
Return an AmusingPreciseNumber that is the absolute value of numb and leave numb unchanged.

Assignment 3 (100 points) An "Amusing" Precise Number PostFix Calculator

The Deque interface allows you to implement both stack and queue data structures. Using an implementation of the List interface in a **HAS-A** relationship, implement the Deque interface. The implementation class name is Deque228. Thus, the Deque228 implementation contains a member variable that is an implementation of the List interface.

Write a main program that uses AmusingPreciseNumber and Deque228 to create a postfix calculator. This program allows the user to input data that contains a sequence of operations and numbers to perform a calculation. The input format is a stream of numbers and operators separated by whitespace characters in operator postfix order. The numbers are in the same format as the AmusingPreciseNumber constructor that takes a String argument. The operators are the following with the obvious meanings: +, -, abs, neg. Any number of abs and neg operators may come after a number. The operators "neg" and "abs" take a single argument. The following examples show an initial expression and how they are evaluated at each step. (The => divides the steps.)

3 2 + neg abs 6 - => 5 neg abs 6 - => -5 abs 6 - => 5 6 - => -1

20 30 - neg => 10

2 5 6 - + => 2 -1 + => 1

Note that in the above example -1 means a negative number, while - with space on each side is the subtraction operator. Using a stack to compute postfix expression will be talked about in lecture, or you may google postfix expressions to learn about them.

The output of the program is the result (in a precise number format) of the calculation on the input that was read from the standard input.

Input to the program is given through the standard input. Output is to standard output. If the input contains an illegal number, or an illegal expression (e.g., "1 4 + +") the output should print an error message. It is useful to use I/O redirection to test your program.

Testing

You are strongly encouraged to test all your classes with Junit, and sharing these tests are encouraged provided they do not contain code that is part of the assignment.

Packaging

Place all class files in the package "cs228hw2" If you write your own Junit tests, please place them in

the package cs228hw2.tests. Like assignment 1, the cs228hw2 directory is a subdirectory of “hw2”.

Please do not wait until the week before it is due to start the assignment.