**SI Linked Lists**

**General:**

Draw the variables, references, and objects that exist after the following code executes. Draw node objects as shown below and boxes for variables. The example has all instance variables set to null. The example does not show any of the variables that actually refer to the node object. You must show all variables and their references in your drawing. Use arrows to show references and a forward slash to indicate variables that storenull. Assume the node class is the doubly linked node from the linked list assignment and that the fields of the class are all public.

prev data next

DoubleListNode<Integer> n1 = new DoubleListNode<>(null, new Integer(7), null);

DoubleListNode<Object> n2 = new DoubleListNode<>(null, null, n1);

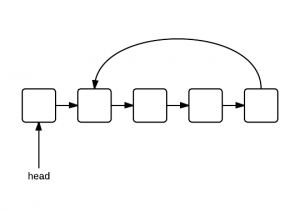
n1.prev = n2;

n1.prev.next.next = n1;

n2.data = n1.prev.next;

n1 = n2;

(From Interview Question) Detect if a linked list is a cycle (picture shown below to illustrate an example). The class is



public class LinkedList<E> {

// refers to first node in the chain of nodes.

private Node<E> first;

// No other instance variables

// The nested Node class.

private static class Node<E> {

private E data;

private Node<E> next;

\\ getters and setters

}

}

Make sure you are not just checking for the same value since you can have multiple values in a linked list. This method is in the LinkedList class. The first node is not null. The time order should be O(N) where N is the number of unique nodes in the list. You can do this in constant space order but do not try to think of that solution first.

public boolean hasCycle() {

(From Fall 2016 midterm 2) Complete the removeNum(int start, int number) instance method for the LinkedList314class. The method removes number elements from the linked list starting at position start. Note, it may not be possible to remove the given number of elements if there are not sufficient elements in list from the given starting position. The precondition requires start to be inbounds, but only requires number to be >= 0. The method returns the actual number of items removed. See the example below.

* You may not use any other methods in the LinkedList314 class unless you implement them yourself as a part of your solution.
* The LinkedList314 class uses singly linked nodes.
* The list has references to the first node in the linked structure of nodes.
* When the list is empty, first is set to null.
* If the list is not empty the last node in the list has its next reference set to null.
* You may use the nested Node class.
* You may not use any other Java classes or native arrays.

public class LinkedList314<E> {

// refers to first node in the chain of nodes.

private Node<E> first;

// No other instance variables

// The nested Node class.

private static class Node<E> {

private E data;

private Node<E> next;

\\ getters and setters

}

}

Examples of calls to int removeNum(int start, int number).

[4].removeNum(0, 1) -> returns 1, list becomes []

[4].removeNum(0, 0) -> returns 0, list remains [4]

[4, 3, 0, 5, 7].removeNum(0, 2) -> returns 2, list becomes [0, 5, 7]

[4, 3, 0, 5, 7].removeNum(3, 2) -> returns 2, list becomes [4, 3, 0]

[4, 3, 0, 5, 7].removeNum(3, 5) -> returns 2, list becomes [4, 3, 0]

/\* pre: start is a valid position in this list, number >= 0

post: per the problem description \*/

public int removeNum (int start, int number) {