CS1102: Data Structures and Algorithms

Tutorial 3 - Linked Lists

Week of 8/2/2010

1. The following four classes are available to you. The `ListNode` class represents a linked list node, while the `DListNode` class represents a doubly linked list node. Linked lists and doubly linked lists are represented by the `BasicLinkedList` and `DoublyLinkedList` classes respectively.

```
class ListNode<E> {
  private E element;
  private ListNode<E> next;

  public ListNode(E item, ListNode<E> next) { ... }
  public E getElement() { ... }
  public void setElement(E item) { ... }
  public ListNode<E> getNext() { ... }
  public void setNext(ListNode<E> next) { ... }
}

class DListNode<E> extends ListNode<E> {
  private DListNode<E> prev;

  public DListNode(E item, DListNode<E> next,
    DListNode<E> prev) { ... }
  public DListNode<E> getPrev() { ... }
  public void setPrev(DListNode<E> prev) { ... }
}

class BasicLinkedList<E> {
  protected int num_nodes = 0;
  protected ListNode<E> head = null;

  public int size() { ... }
  public ListNode<E> getHead() { ... }
  public void addHead(E item) { ... }
  public void deleteHead() { ... }
}

class DoublyLinkedList<E> {
  protected int num_nodes = 0;
  protected DListNode<E> head = null;

  public int size() { ... }
  public DListNode<E> getHead() { ... }
  public void addHead(E item) { ... }
  public void deleteHead() { ... }
}
```

a. For the `BasicLinkedList` class, write a new method: `public void switchWithNext(ListNode<E> node)` that switches the position of `node` with the node at `node.getNext()`. If switching is not applicable, the method does nothing. Similarly, write a new method: `public void switchWithNext(DListNode<E> node)` for the `DoublyLinkedList` class.

b. The following method is added to the `BasicLinkedList` class:

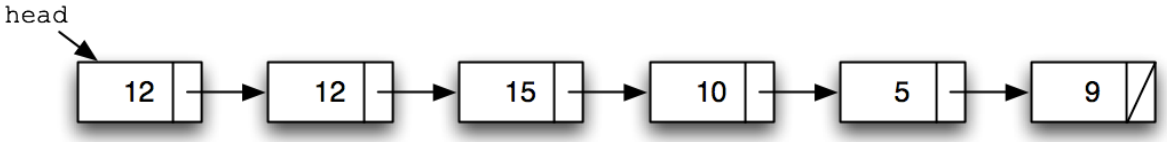
```
public boolean mysteryMethod() {
  ListNode<E> current = getHead();
  boolean flag = false;
  if(current == null) return flag;
  while(current.getNext() != null) {
    if(current.getElement().compareTo(
        current.getNext().getElement()) > 0) {
      switchWithNext(current);
      flag = true;
    } else {
      current = current.getNext();
    }
  }
  return flag;
}
```

The declaration of the class is also changed to:

```
class BasicLinkedList<E extends Comparable<? super E>>
```

The linked list `myLinkedList` below is an instance of the `BasicLinkedList<Integer>` class. Draw what it would look like after calling `myLinkedList.mysteryMethod()`. Then in one sentence, explain what `myLinkedList.mysteryMethod()` does to `myLinkedList`.

```
BasicLinkedList<Integer> myLinkedList
```

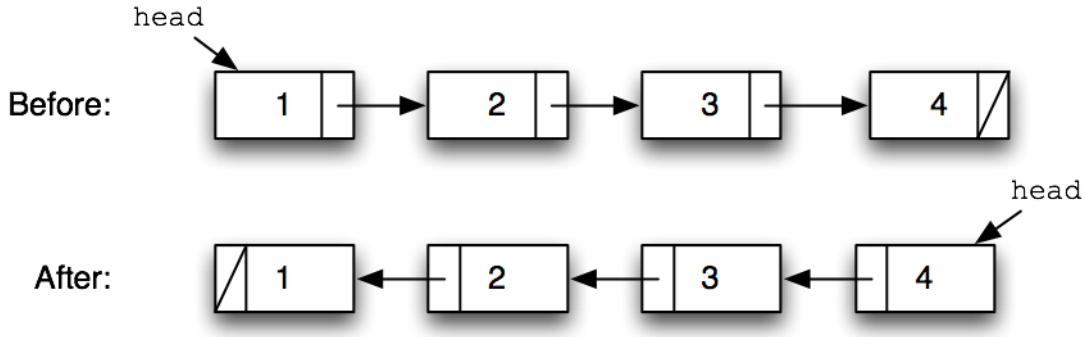


c. The following method is added to the `BasicLinkedList` class in (b):

```
public void mysteriousMethod() {
  boolean flag = mysteryMethod();
  while(flag == true) {
    flag = mysteryMethod();
  }
}
```

In one sentence, explain what `myLinkedList.mysteriousMethod()` does to `myLinkedList`.

2. Write a method to reverse a linear linked list as illustrated below. Your method should not instantiate any new nodes. Your method should be included in the `BasicLinkedList` class (question 1).




```
class BasicLinkedList<E> {
  // Other methods as defined in question 1

  public void reverse() {
    // Your code here
  }
}
```

3. (Circular linked list) Using what we have explored in the lectures, write a java program to implement a circular linked list class which extends BasicLinkedList class with a tail pointer. You should write a constructor which creates a circular linked list from a given BasicLinkedList. You should also write an addTail ( ) method, which adds a new node as the last node in the list. And lastly you should write a toString( ) method, which go through the circular list and returns the String form of every element. Please be noted that class E contains or inherits a toString( ) method.

```
public class CircularList<E> extends BasicLinkedList<E>{
   protected ListNode<E> tail;

   public CircularList(){
         super();
         tail=null;
   }

   public CircularList(BasicLinkedList<E> list){
   // Your code here
   }

   public addTail(E item){
   // Your code here
   }

   public String toString(){
   // Your code here
   }
}
```

4. The following class extends the `BasicLinkedList<E>` class from question 1:

```
class TailedLinkedList<E> extends BasicLinkedList<E> {
  protected ListNode<E> tail = null;

  public ListNode<E> getTail() { ... }
  public void addTail(E item) { ... }
  public void deleteTail() { ... }
  public void addHead(E item) { ... } // Overridden to handle
  public void deleteHead() { ... }    // tail pointer correctly.
  public void insertAfter(ListNode<E> node, E item) { ... }
}
```

Given an instance of a `TailedLinkedList<Integer>` class from above, write a method that finds the largest k Integer values in that linked list. You may assume that k will always be given as a positive integer. Your method should return a linked list (`TailedLinkedList<Integer>`) of the largest k Integer values. This linked list must be sorted in descending order (the head node stores the largest Integer value). If the size of the given linked list is less than k, your method should return a new linked list of the same size with all the Integer values from the given linked list sorted in descending order. You are not allowed to use arrays in your method. You should use the `TailedLinkedList<Integer>` class instead. Also note that the original linked list must not be destroyed by your method.

```
class LargestIntegers {
  public static TailedLinkedList<Integer> findKLargest(
      TailedLinkedList<Integer> myLinkedList, int k) {
    // Your code here
  }
}
```