APE Lab – Linked Lists and Recursion

The linked list material is spread through several folders, so that difficulties in getting any single method to work will not prevent testing your knowledge of other areas. This will be a non-circular linked list with a dummy header node. (Because of the hierarchical structure, ***all*** of the linked list subclasses will also be non-circular lists with a dummy header node.) Go into each of the folders shown (which constitute a long hierarchy of superclass/subclass files) and make the indicated changes. **In none of these are you allowed to change the driver file.** You will be given working versions of the .class files from the superclass(es) in each of the subclasses.

LinkedList0
Write the **isEmpty(), clear()**, **addFirst()** and **listForward()** methods for **LinkedList0.java**. Check the output file **LinkedList0.txt** to see what should be produced from running **Driver0**, including the format for **listForward()**. If you wish, you may add helper methods to accomplish the specified public behaviors. **Note:** if you wish to use accessor methods (getData/setData, getNext/setNext), you may add them to the Node class and they will be available in the LinkedList0$Node.class files in the subclasses derived from the class LinkedList0 in the remaining parts of this exam.

LinkedList1
Write the **Comparable get(int index)** method for **LinkedList1**. If **index** is invalid as an index [less than zero or greater than or equal to the list size], print an error diagnostic and return a null reference. Otherwise return the reference to the Comparable that occurs at the indicated **index**. Do **NOT** delete **LinkedList0$Node.class,** or **LinkedList0.class** from the folder. Check the output file **LinkedList1.txt** to see what should be produced from running **Driver1**.

LinkedList2
Write the **void addLast(Comparable element)** method for **LinkedList2**. This is to add the indicated object (**element**) so that it is the last element within **LinkedList2**. [Note that this implementation of a linked list does **not** have a tail reference.] Do **NOT** delete **LinkedList0$Node.class, LinkedList0.class**, or **LinkedList1.class** from the folder. Check the output file **LinkedList2.txt** to see what should be produced from running **Driver2**.

LinkedList3 — **Order Relationships**
Write the **void addOrdered(Comparable element)** method for **LinkedList3**. This is to add the indicated Comparable object (**element**) into its proper position within **LinkedList3**. Do **NOT** delete **LinkedList0$Node.class, LinkedList0.class**, **LinkedList1.class**, or **LinkedList2.class** from the folder. Check the output file **LinkedList3.txt** to see what should be produced from running **Driver3**.

LinkedList4
Write the **boolean remove(Comparable element)** method for **LinkedList4**. This is to find the first occurrence of **element** within the list, remove that node and return true. If it fails to find **element**, it returns a false. Do **NOT** delete **LinkedList0$Node.class, LinkedList0.class, LinkedList1.class**, **LinkedList2.class**, or **LinkedList3.class** from the folder. Check the output file **LinkedList4.txt** to see what should be produced from running **Driver4**.

LinkedList5 — **Recursive Component**

Write the **listReverse()** method for **LinkedList5.java**. This method should print the contents of the list in REVERSE order. You must utilize recursion to accomplish this task. You are welcome to write one or more helper methods to aid your recursion. Do **NOT** delete **LinkedList0$Node.class, LinkedList0.class, LinkedList1.class**, **LinkedList2.class, LinkedList3.class**, or **LinkedList4.class** from the folder. Check the output file **LinkedList5.txt** to see what should be produced from running **Driver5**.