CS 284 C: Quiz 4
Spring 2020
Time: 15 minutes

Student Name:
Honor Pledge:

Exercise 1 (*5 points*)

Write the method SingleLinkedList⟨E⟩ stutterNL() that repeats each element in the single linked list i times, where i is the index of the element, starting from 1. Eg. If this.head = [1, 3, 5], it should replace this.head with [1, 3, 3, 5, 5, 5].

```java
import java.util.ArrayList;

public class SingleLinkedList<E> {

  private static class Node<E> {
      private E data;
      private Node<E> next;
      /** Creates a new node with a null next field
          @param dataItem  The data stored
      */

      private Node(E dataItem) {
        data = dataItem;
        next = null;
      }
  }

  Node<E> head;

    /**
   * return a linked linked consisting of node copied n times
   * @param node
   * @param n
   * @return the head and tail of the output list
   */
  public ArrayList<Node<E>> sub_copy(Node<E> node, int n) {
    Node<E> head = node;
    for (int j = 0; j < n - 1; j++) {
      Node<E> copy_node = new Node<E>(node.data);
      node.next = copy_node;
      node = copy_node;
    }
    ArrayList<Node<E>> ret_array = new ArrayList<Node<E>>();
    ret_array.add(head);
    ret_array.add(node);
    return ret_array;
  }

  /**
   * repeats each element in the single linked list i times, where i is the index
   of the element, starting from 1. Eg. if this.head = [1, 3, 5], it should replace
   this.head with [1, 3, 3, 5, 5, 5].
   *
   */
  public void stutterNL(){

    if (this.head != null) {
      Node<E> node = this.head;
      Node<E> node_next = node.next;
      ArrayList<Node<E>> sub_list = this.sub_copy(node, 1);
      Node<E> all_head = sub_list.get(0);
      Node<E> new_tail = sub_list.get(1);
      int counter = 2;
      while (node_next != null) {
        node = node_next;
        node_next = node.next;
        sub_list = this.sub_copy(node, counter);
        counter ++;
        Node<E> new_head = sub_list.get(0);
        new_tail.next = new_head;
        new_tail = sub_list.get(1);
      }
      this.head = all_head;
     }

   }
}
```

Exercise 2 (*5 points*)

Write a method `public void compress(Node⟨E⟩ node_head)` that compresses a list by counting repetitions of adjacent elements, where the head of the input list is node_head. For example, the result of applying this operation to [4,4,4,2,3,3,2,2,2,1,1] should be [(4,3),(2,1),(3,2),(2,3),(1,2)]. At the end of the function, set `head` as the head of the compressed list.

Hint: Consider separate cases for when the list is empty, a singleton, or has two or more elements.

```
public class PairLinkedList<E> {

  private static class Node<E> {
      private E data;
      private Node<E> next;
      /** Creates a new node with a null next field
          @param dataItem  The data stored
      */

      private Node(E dataItem) {
        data = dataItem;
        next = null;
      }
  }

  private static class Pair<E, Integer> {
      private E data;
      private Integer copy_count;
      private Pair<E, Integer> next;
      /** Creates a new pair with a null next field
          @param dataItem  The data stored
      */

      private Pair(E dataItem) {
        data = dataItem;
        next = null;
      }

      /**
       * set the number of copies as copy
       * @param copy
       */
      private void set_copy(Integer copy) {
        copy_count = copy;
      }
  }

  Pair<E, Integer> head;

  /**
   * compresses a list by counting repetitions of adjacent elements, where the head of
   the input list is node_head. Eg, the result of applying this operation to
   [4,4,4,2,3,3,2,2,2,1,1] should be [(4,3),(2,1),(3,2),(2,3),(1,2)]
   * @param node_head
   */
  public void compress(Node<E> node_head){

    if (node_head != null) {
      Node<E> node = node_head;
      Pair<E, Integer> current_node = new Pair(node.data);
      Pair<E, Integer> new_head = current_node;
      int node_count = 1;
      E prev_Data = node.data;
      while (node.next != null) {
        node = node.next;
        if (node.data.equals(prev_Data) == false) {
          current_node.set_copy(node_count);
          Pair<E, Integer> next_node = new Pair(node.data);
          current_node.next = next_node;
          current_node = next_node;
          node_count = 1;
        }
        else {
          node_count += 1;
        }
        prev_Data = node.data;
      }
      current_node.set_copy(node_count);
      this.head = new_head;
    }
  }
```