CS 2383: Data Structures and Algorithms

Assignment 1: Assignment Title (if any)

Student Name: Yulong Wang Student Number: 3713596

[Mandatory] Declaration: "I warrant that this is my own work."

Signed by

[Optional] "I hereby give my permission for this work to be used (with my name and identifying information removed) for UNB Faculty of Computer Science program accreditation purposes."

Signed by

Q1:

```
public class AbsoluteProgression extends Progression{
    private long previous;
    AbsoluteProgression(long first, long second){
        current = second;
        first = previous;
    }

    AbsoluteProgression(){
        current = 200;
        previous = 2;
    }

    @Override
    protected void advance() {
        long temp = current;
        current =Math.abs(current - previous);
        previous = temp;
    }

    @Override
    public void printProgression(int n) {
        System.out.print(previous + " ");
        super.printProgression(n-1);
    }

    public static void main(String[] args) {
        AbsoluteProgression temp = new AbsoluteProgression();
        temp.printProgression(5);
    }


}
```

Q2:

```java
import java.util.Arrays;
import java.util.Iterator;

public class SinglyLinkedList<E> {
    //---------------- nested Node class ----------------
    private static class Node<E> {
        private E element; // reference to the element stored at this node
        private Node<E> next; // reference to the subsequent node in the list

        public Node(E e, Node<E> n) {
            element = e;
            next = n;
        }

        public E getElement() {
            return element;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> n) {
            next = n;
        }
    } //----------- end of nested Node class -----------

    // instance variables of the SinglyLinkedList
    private Node<E> head = null; // head node of the list (or null if empty)
    private Node<E> tail = null; // last node of the list (or null if empty)
    private int size = 0; // number of nodes in the list

    // construct an initially empty list
    public SinglyLinkedList() {
    }

    // construct the list from an array of generic type (**updated from
earlier version**); incomplete
    public SinglyLinkedList(E[] elements) {
        Iterator<E> itr = Arrays.stream(elements).iterator();
        if(itr.hasNext()){
            head = new Node<>(itr.next(), null);
            Node<E> curr = head;
            size ++;
            while(itr.hasNext()){
                curr.next = new Node<>(itr.next(), null);
                curr = curr.next;
                size++;
            }
            tail = curr;
        }
    }

    // add an element as the first node
    public void addFirst(E e) {
```

```java
        head = new Node<>(e, head);
        if (size == 0)
            tail = head;
        size++;
    }

    // reverse the list; incomplete
    public void reverse() {
        Node<E> pre = null;
        Node<E> curr = head;
        Node<E> next = head;

        while(curr != null){
            next = next.next;
            curr.next = pre;
            pre = curr;
            curr = next;
        }
        curr = tail;
        tail = head;
        head = curr;
    }

    // print elements in this list; incomplete
    public void printElements() {
        Node<E> curr = head;
        while(curr!=null){
            System.out.print(curr.element + " ");
            curr = curr.next;
        }
    }

    // testing method; incomplete
    public static void main(String[] args) {
        Integer[] testData = {3, 2, 7, 0, 1, 9, 4};

        SinglyLinkedList<Integer> testList = new
SinglyLinkedList<Integer>(testData);
        // Think about how to create the following singly linked list
        testList.printElements();
        System.out.println();
        // Reverse list and print again
        testList.reverse();
        testList.printElements();
    }
}
```

Q3:

```java
public class IntegerRearranger {
    public static int[] rearranger(int [] nums){
        if(nums.length < 2){
            return nums;
        }
        return rearrangerHelper(nums, 0);
    }

    private static int[] rearrangerHelper(int[] nums, int boundary){
        if(boundary == nums.length){
            return nums;
        }
        int last = nums[nums.length-1];
        if(last%2 == 0){
            //even
            for(int i =nums.length-1; i>0 ;i--){
                nums[i] = nums[i-1];
            }
            nums[0] = last;
        }else{
            //odd
            for(int i =nums.length-1; i>boundary ;i--){
                nums[i] = nums[i-1];
            }
            nums[boundary] = last;
        }
        boundary += 1;
        return rearrangerHelper(nums,boundary);
    }



    public static void main(String[] args) {
        int[] data = {1, 4, 7, 2, 10, 5};
        int[] rearranged = rearranger(data);
        for(int i : rearranged){
            System.out.print(i + " ");
        }
    }
}
```