Your login name: cs61bl-_____

Problem 1 (7 points)

Given below are Counter and ModNCounter classes similar to those described in lab exercises.

```
public class Counter {
   protected int myCount;
   public Counter ( ) {
      myCount = 0;
   }
   public void increment ( ) {
      myCount++;
   }
   public void reset ( ) {
      myCount = 0;
   }
   public int value ( ) {
      return myCount;
   }
   public String toString ( ) {
      return "" + value ( );
   }
}
```

```
public class ModNCounter extends Counter {
   private int myN;
   public ModNCounter (int n) {
      myN = n;
   }
   public int value ( ) {
      return myCount % myN;
   }
}
```

Part a

Consider the following program segment:

```
ModNCounter ctr = new ModNCounter (2);
for (int k=1; k<=5; k++) {
   ctr.increment ( );
}
System.out.println (ctr);
```

What gets printed, 5, 1, or something else? Briefly explain your answer.

Part b

Provide a definition for a class named SeasonProgression that is derived from class ModNCounter via inheritance. A SeasonProgression object will essentially act like a mod 4 counter. A call to its toString method returns one of the strings “spring”, “summer”, “autumn”, or “winter”; the string cycles from “winter” to “spring”. The program segment below should produce the indicated output.

program segment

```
SeasonProgression s = new SeasonProgression ( );
s.reset ( );
System.out.println (s);
for (int k=1; k<=5; k++) {
   s.increment ( );
   System.out.println (s);
}
```

desired output

```
spring
summer
autumn
winter
spring
summer
```

Don’t define any new instance variables in your SeasonProgression class, and rely as much as possible on the methods of ModNCounter.

Problem 2 (15 points)

Both parts of this question involve a class SeqArray that represents an array of IntSequence objects. Every cell in the array is assumed to contain an IntSequence object. Outlines of the class definitions for SeqArray and IntSequence appear in a separate handout; the IntSequence class is similar to what you encountered in lab.

Part a

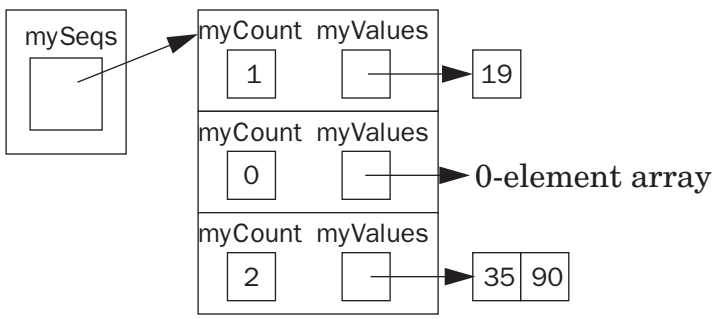
On the next page, fill in the blanks in the body of the SeqArray constructor. Assume that the input has the following form.

- The first line of the input contains the number of IntSequence objects to be stored.
- Groups of input values follow, with the kth group specifying the size and elements of the kth IntSequence. In each group, the size is specified first, followed by the elements of the IntSequence.

Here is an example.

input *resulting* SeqArray

```
3
1
19
0
2
35
90
```



You won't need to fill in every blank.

Space for your answer to problem 2, part a

```
public class SeqArray {
   private IntSequence [ ] mySeqs;
   // Return the next integer in the input (assuming integers are provided one per line).
   private static int nextInt (BufferedReader in) throws Exception {
      return Integer.parseInt (in.readLine ( ));
   }
   // Initialize this SeqArray by reading from the given input source.
   public SeqArray (BufferedReader in) throws Exception {


      int arrayLen = nextInt (in);


      for (int k1=0; k1<arrayLen; k1++)


         int seqLen = nextInt (in)


         for (int k2=0; k2<seqLen; k2++) {


            int x = nextInt (in);


         }


      }


   }
```

A CS 61BL student supplies the following code for the SeqArray ElemEnumeration class.

```
public class ElemEnumeration {

   private int index;
   private IntSequence.IntEnumeration enum;

   // Initialize an iterator for all integers in all the sequences.

   public ElemEnumeration ( ) {
      index = 0;
      if (hasMoreElements ( )) {
         enum = mySeqs[0].elements ( );
      }
   }

   // Return true if there are more integers left to return in any of the sequences;
   // return false if we've gone through all the integers in all the sequences.

   public boolean hasMoreElements ( ) {
      return index < mySeqs.length;
   }

   // Return the next integer.
   // If we run out of integers in one sequence, go on to the next.
   // Precondition: hasMoreElements ( ).

   public int nextElement ( ) {
      int x = enum.nextElement ( );
      if (!enum.hasMoreElements ( )) {
         index++;
         if (hasMoreElements ( )) {
            enum = mySeqs[index].elements ( );
         }
      }
      return x;
   }
}
```

The code has a bug.

Part b

Give an example of an array of sequences for which the iterator fails to work correctly. Also explain in detail what behavior results from running the iterator on your example: if it crashes, say where and why; if it returns an incorrect value, indicate what value is returned and what the correct value to return would be.

Part c

Fix the iterator, making as few changes to existing code as possible. Here's the code for your reference.

```
public class ElemEnumeration {

   private int index;
   private IntSequence.IntEnumeration enum;

   public ElemEnumeration ( ) {
      index = 0;
      if (hasMoreElements ( )) {
         enum = mySeqs[0].elements ( );
      }
   }

   public boolean hasMoreElements ( ) {
      return index < mySeqs.length;
   }

   public int nextElement ( ) {
      int x = enum.nextElement ( );
      if (!enum.hasMoreElements ( )) {
         index++;
         if (hasMoreElements ( )) {
            enum = mySeqs[index].elements ( );
         }
      }
      return x;
   }
}
```

Problem 3 (8 points)

Write a SchemeList method that, given a nonnegative int k, rotates the nodes in this list by k positions. That is, it should unlink the first k nodes in the list and move them to the end of the list, as shown below. Assume that, if N is the number of elements in this list, then k < N. Don't create any new ConsNodes or use any other methods other than those you define yourself.

Examples:

k	*represented list before rotation*	*represented list after rotation*
0	(A B C D)	(A B C D)
2	(A B C D)	(C D A B)
3	(A B C D)	(D A B C)

```
public class SchemeList {
   private ConsNode myHead; // pointer to the first node in a nonempty list
   private ConsNode myTail; // pointer to the last node in a nonempty list
   private class ConsNode {
      public Object myCar;
      public ConsNode myCdr;
   }
   public void rotate (int k) {
```

Outline of the SeqArray class

This class represents an array of IntSequence objects.

```
public class SeqArray {
   private IntSequence [ ] mySeqs;
   // Initialize this SeqArray by reading from the given input source.
   public SeqArray (BufferedReader in) throws Exception {
      ...
   }
   // Return the next integer in the input (assuming integers are provided one per line).
   private static int nextInt (BufferedReader in) throws Exception {
      return Integer.parseInt (in.readLine ( ));
   }
   // Return an enumeration of all the elements of all the stored sequences.
   public ElemEnumeration allElems ( ) {
      return new ElemEnumeration ( );
   }
   public class ElemEnumeration {
      ...
   // Initialize an iterator for all integers in all the sequences.
      public ElemEnumeration ( ) {
         ...
      }
      // Return true if there are more integers left to return in any of the sequences;
      // return false if we've gone through all the integers in all the sequences.
      public boolean hasMoreElements ( ) {
         ...
      }
      // Return the next integer.
      // If we run out of integers in one sequence, go on to the next.
      // Precondition: hasMoreElements ( ).
      public int nextElement ( ) {
         ...
      }
   }
}
```

Outline of the IntSequence class

```
public class IntSequence {
   private int [ ] myValues;
   private int myCount;
   // Initialize a sequence to hold at most the given number of integers.
   public IntSequence (int capacity) {
      ...
   }
   // Return a String representation of the sequence.
   public String toString ( ) {
      ...
   }
   // Return true if the sequence is empty; return false otherwise.
   public boolean isEmpty ( ) {
      ...
   }
   // Add the argument integer to the end of the sequence.
   public void addToSequence (int toAdd) {
      ...
   }
   // Return an iterator for integers in the sequence.
   public IntEnumeration elements ( ) {
      return new IntEnumeration ( );
   }
   public class IntEnumeration {
      // Initialize an iterator for integers of the sequence.
      public IntEnumeration ( ) {
         ...
      }
      // Return true if there are more integers left to return;
      // return false if we've gone through all the integers in the sequence.
      public boolean hasMoreElements ( ) {
         ...
      }
      // Return the next integer in the sequence. Precondition: hasMoreElements ( ).
      public int nextElement ( ) {
         ...
   }
}
```