Problem 1

Part a (2 points)

The given program segment prints 1. The call to println results in a call to toString, which calls the value method. The ModNCounter object uses the Counter toString method. However, it uses the ModNCounter value method, even though toString appears only in the Counter class, because a ModNCounter will always use its own version of an overridden method.

Part b (5 points)

Here's the definition of the SeasonProgression class.

```
public class SeasonProgression extends ModNCounter {

   public SeasonProgression ( ) {
      super (4);
   }

   public String toString ( ) {
      int n = value ( ); // super.value ( ) works too.
      if (n == 0) {
         return "spring";
      } else if (n == 1) {
         return "summer";
      } else if (n == 2) {
         return "autumn";
      } else {
         return "winter";
      }
   }
}
```

You could have used a switch to select the appropriate string. (We didn't cover switch, but you may have learned about it when studying another C-based language.) Another possibility is to access an array:

```
public String toString ( ) {
   String [ ] seasonNames = {"spring", "summer", "autumn", "winter"};
   return seasonNames[value ( )];
}
```

Problem 2

Part a (6 points)

There were three operations to be supplied to the given framework:

- the initialization of the mySeqs array;
- the initialization of each IntSequence element of the mySeqs array;
- the inclusion of each relevant input value in the appropriate IntSequence.

Here's the code, with the statements you were to add in boldface..

```
public SeqArray (BufferedReader in) throws Exception {
   int n = nextInt (in);
   mySeqs = new IntSequence [arrayLen];
   for (int k1=0; k1<arrayLen; k1++) {
      int seqLen = nextInt (in);
      mySeqs[k1] = new IntSequence (m);
      for (int k2=0; k2<seqLen; k2++) {
         int x = nextInt (in);
         mySeqs[k1].addToSequence (x);
      }
   }
}
```

Part b (3 points)

The given code fails to handle a 0-length IntSequence correctly. Suppose, for example, that the first IntSequence is empty. The variable enum is correctly initialized to an enumeration of the empty sequence's elements. However, the hasMoreElements method allows us to call nextElement, which immediately tries to get the next IntSequence element, which doesn't exist. (An ArrayIndexOutOfBoundsException probably results.)

Part c (6 points)

Our intended solution was to maintain the following invariant property:

- enum.nextElement ( ) is the next element to be returned in the SeqArray enumeration.

This would be done by changing the constructor and nextElement as follows.

```
public ElemEnumeration ( ) {
   index = 0;
   advance ( );
}
private void advance ( ) {
   while (index < mySeqs.length) {
      enum = mySeqs[index].elements ( );
      if (enum.hasMoreElements ( )) {
         return;
      }
      index++;
   }
}
```

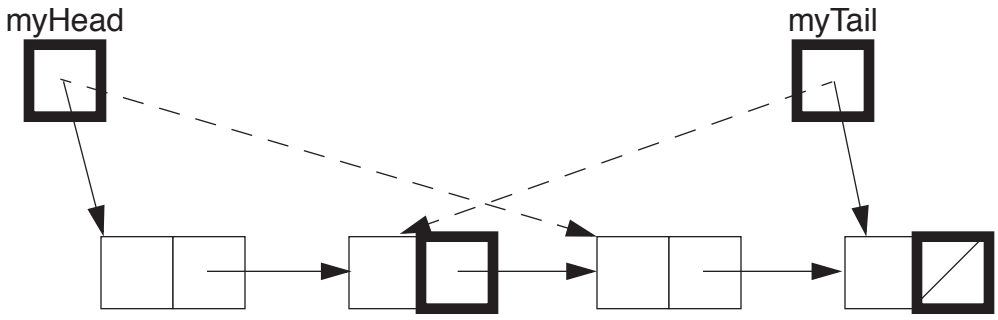
```
public int nextElement ( ) {
   int x = enum.nextElement ( );
   if (!enum.hasMoreElements ( )) {
      index++;
      advance ( );
   }
   return x;
}
```

A variation of the advance method checks mySeqs[index].isEmpty before calling the IntSequence elements method.

A loop or recursive call is necessary to avoid the situation of consecutive empty IntSequences. (You encountered this situation in other contexts in earlier lab exercises.)

Problem 3

This was worth 8 points. The diagram below portrays the situation at the start of the call rotate (2). The boxes outline in bold are to be updated. Dotted lines indicate the new values of myHead and myTail; the myCdr field of the last ConsNode should be pointed at the head of the list, and the myCdr field of the second ConsNode should be set to null.



Here are two implementations, one iterative and one recursive. The recursive version does k rotations rather than just one.

```
public void rotate (int k) {
   if (k>0) {
      // Find node before the new head.
      ConsNode newTail = myHead;
      for (int k2=1; k2<k; k2++) {
         p = p.myCdr;
      }
      ConsNode temp = myHead;
      myHead = p.myCdr;
      myTail.myCdr = temp;
      myTail = p;
      p.myCdr = null;
   }
}
```

```
public void rotate (int k) {
   if (k>0) {
      myTail.myCdr = myHead;
      myTail = myHead;
      myHead = myHead.myCdr;
      myTail.myCdr = null;
      rotate (k-1);
   }
}
```