Question 7. (21 points) Inheritance. Consider the following class definitions:

```
public class Insect {
   public void noise() { System.out.println("scratch"); }
   public void hello() { System.out.println("hi"); }
}

public class Ant extends Insect {
   public void noise() { System.out.println("dig dig"); }
}

public class Bug extends Insect  {
   public void   fly()   { System.out.println("flap flap"); }
   public void   noise() { System.out.println("Bug says " + what()); }
   public String what()  { return "buzz"; }
}

public class Fly extends Bug {
   public void hello() { System.out.println("howdy"); }
   public void fly()   {
      System.out.println("fly");
      super.fly();
   }
}

public class Bee extends Bug {
   public void   sting() { System.out.println("zap!"); }
   public String what()  { return "honey";  }
}
```

Answer the questions about these classes on the next page. You may remove this page from the exam for reference if you wish.

Question 7. (cont.) For each of the following code sequences write down the output produced, if any, if the code executes successfully. If there is either a compile-time or run-time error that prevents successful execution, give a concise description of the problem. If there are errors, they're not trivial ones like incorrect punctuation.

(a)
```
Insect em = new Ant();
em.noise();
```

dig dig

(b)
```
Bug black = new Fly();
black.fly();
```

fly
flap flap

(c)
```
Bug bunny = new Bug();
bunny.noise();
```

Bug says buzz

(d)
```
Bee b = new Bee();
b.sting();
```

zap!

(e)
```
Fly house = new Bug();
house.fly();
```

Compiler error: incompatible types. Can't assign a value of type Bug to a variable of type Fly.

(f)
```
Insect thing = new Bug();
((Bee) thing).sting();
```

Runtime ClassCastException error. Can't cast value of type Bug to type Bee.

(g)
```
Bug bumble = new Bee();
bumble.noise();
```

Bug says honey

Question 8. (24 points) Suppose we have a class that stores a set of integer values with no duplicates. In this problem the values are stored in a linked list, and the values in the list are kept in sorted, ascending order.

Complete the definition of the method `containsAll` on the next page so it returns true if all of the items in a second set (`other`) are also contained in the current set. The method should return false if some item is found in the other set that is not contained in the current one. For example, suppose the current set contains {3, 6, 12, 19, 25}. If set `x` contains the values {3, 12, 19}, then `containsAll(x)` should return `true`. If set `y` contains the values {6, 19, 42}, then `containsAll(y)` should return false. If the other set is empty, then `containsAll` should return `true`, since there are no items in the other (empty) set that are missing from the current one.

You may define additional private methods and simple variables as needed, but you may not create additional collections, nodes, or other objects. You may not modify either of the sets involved.

For full credit, your method should run in no more than $O(n+m)$ time where n and m are the number of items in the two sets.

The first part of the class definition is given below.

```
public class IntListSet {

   // list nodes
   private class IntListNode {
      public int data;           // value in this node
      public IntListNode next;   // next node or null if none
   }

   // list of values in this IntListSet, stored in ascending order
   // with no duplicates.  Null if the set is empty.
   private IntListNode items;

   // construct empty IntListSet
   public IntListSet() { items = null; }
```

Complete the `containsAll` method for this class in the space provided on the next page.

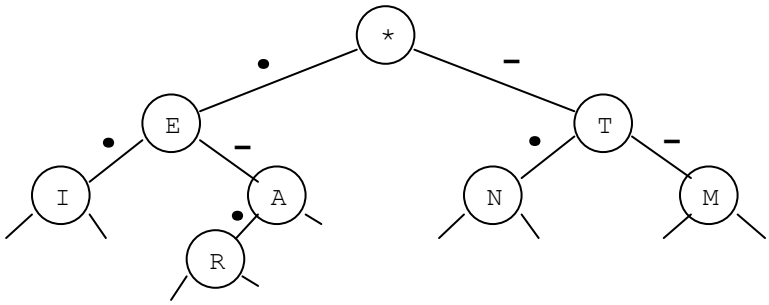
Feel free to remove this page from the exam for reference if you wish.

Question 8. (cont). Complete the method definition below (you may add additional methods if you need them).

```
// Return true if every item in the IntListSet other is also
// contained in this IntListSet.  Return false if some item found
// in IntListSet other is not found in this IntListSet.
public boolean containsAll(IntListSet other) {

   // Advance through lists comparing items and quit when we reach
   // the end of either list or if we find an item in the other list
   // that is not contained in this one.

   IntListNode p = this.items;
   IntListNode q = other.items;

   while (p != null && q != null) {
      // inv: have not passed the end of either list of values
      if (p.data < q.data) {
         // item in this set not found in other.  skip it.
         p = p.next;
      } else if (p.data == q.data) {
         // item found in both sets.  skip both copies.
         p = p.next;
         q = q.next;
      } else { // p.data > q.data.
               // item in other does not appear in this set.
         return false;
      }
   }

   // Here either p==null or q==null or both.
   // If q != null then there were additional item(s) in other with
   // value(s) greater than the last matching value in this set.

   return q == null;

}
```

Question 9. (24 points) Morse code is a method for encoding letters, digits, and other characters as a sequence of dots and dashes. As with Huffman codes, the codes assigned to frequently used characters are shorter than the codes for infrequent ones. A few examples: the code for `A` is ·−, `E` is · (a single dot), `T` is a single dash −, `F` is ··−·, and `R` is ·−·.

We can use a binary tree to represent Morse code and to decode individual letters. We will arbitrarily decide that the left subtree of a node is the one reached by a dot (·) and the right subtree is reached by a dash (−). With those conventions, the top part of the binary tree for Morse code looks like this:



Looking at the tree we can see that a dot represents `E`, the sequence dash dot is `N` (right then left from the root), and dot dash dot is `R` (left, right, left). Unlike Huffman codes the letters may appear anywhere in the tree, not just in the leaf nodes.

For this problem, complete the definition of method `decode` in class `MorseCode` on the next page so that it returns the character whose Morse code is given by the `String` argument `code`. For instance, if the argument `code` is the string "`..`", method decode should return `I`, which will be the character found in the node reached by starting at the root and following the left (dot) links. If the argument is "`.-.`", the result would be `R`.

You may assume that the `code` argument is valid and there is a corresponding node in the tree with that code, and you should assume that the full tree has been initialized properly to contain all of the possible Morse code sequences and characters.

For full credit, your tree traversal must use recursion to descend the tree. You may add additional private methods and simple variables as needed, but you may not create additional collections, tree nodes, or other objects, and you may not modify the tree.

Complete the method `decode` on the next page. You may remove this page for reference while you're working.

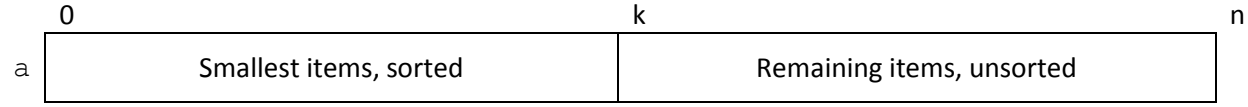
Question 9 (cont.) Complete method `decode` below.

```
public class MorseCode {

   // individual tree node
   private class CodeNode {
      public char ch;            // character in this node
      public CodeNode left;      // subtree reached by a . (dot) from
                                 //    this node
      public CodeNode right;     // subtree reached by a - (dash) from
                                 //    this node
   }

   // root of the code tree
   private CodeNode codeTree;

   // constructor
   public MorseCode(...) { ... }


   // return the character corresponding to the string of dots
   // and dashes given in the argument code.
   public char decode(String code) {
      return decode(code, codeTree);
   }

   // Return the character reached from root r by following the dots
   // and dashes in string s.  If string s is the empty string, return
   // the character contained in node r.
   // pre: r != null.
   public char decode(String s, CodeNode r) {
      if (s.length() == 0) {
         return r.ch;
      } else if (s.charAt(0) == '.') {
         return decode(s.substring(1, s.length()), r.left);
      } else { // s.charAt(0) == '-'
         return decode(s.substring(1, s.length()), r.right);
      }
   }
```

Notes: The problem specified that recursion should be used to descend the tree. It would also be possible to solve the problem with a loop, but some points were deducted from solutions that did that.

Class `String` also provides another version of `substring` with a single argument that returns the remainder of the string beginning at that position. The `substring` operations in the above code could have been written `s.substring(1)` instead of `s.substring(1,s.length())`.

Question 10. (15 points) Sorting. In lecture we discussed some common sorting algorithms. The *selection sort* algorithm can be described by the following picture of a partially sorted array:



That is, when the sort is partially completed, array elements `a[0..k-1]` are sorted (`a[0]` <= `a[1]` <= `...` <= `a[k-1]`), the remaining array elements `a[k..n-1]` are unsorted, and all elements in the sorted part `a[0..k-1]` have values less than or equal to the values in the unsorted part `a[k..n-1]`. Selection sort proceeds by repeating the following steps until no elements remain in the unsorted part:

- Locate the smallest item in the unsorted part `a[k..n-1]`
- Interchange this smallest value with `a[k]` and increase `k` by 1

The following code is a partially completed method to perform a selection sort of an integer array. Complete the method by writing the appropriate code in the box. (Note: The problem can be solved without altering any of the given code, so don't change it.)

```
// sort integer array a in non-decreasing order
public void selectionSort(int[] a) {
   for (int k = 0; k < a.length-1; k++) {
      // Find smallest element in a[k..a.length-1] and swap it with a[k]

      // set minloc = location of smallest value in a[k..a.length-1]
      int minloc = k;
      for (int i = k+1; i < a.length; i++) {
         if (a[i] < a[minloc]) {
            minloc = i;
         }
      }

      // swap a[minloc] and a[k]
      int temp = a[k];
      a[k] = a[minloc];
      a[minloc] = temp;


   }
}
```

Note: A number of solutions rearranged the array in various ways that did, in fact, eventually leave it sorted in non-decreasing order. But solutions that didn't implement the given algorithm roughly as specified lost some points even if the array did wind up sorted somehow.