Throughout this test, assume that the following definition is available. (It is taken from the lecture notes and are duplicated here for your convenience. There is nothing new here.)

```java
class List {
    public Object head; /* First element. */
    public List tail; /* Rest of list. */
    public List(Object head, List tail) {
        this.head = head; this.tail = tail;
    }
}
```

### Problem #1

[6 points] Fill in the bodies of the following functions to agree with their comments. Both of these functions select an evenly spaced subsequence of a list. For example, if L is a list containing the strings

"aardvark", "basilisk", "axolotl", "gibbon", "gnu",
"kumquat", "grapefruit", "kiwi", "poplar"

then both subseq1(L, 1, 3) and subseq2(L, 1, 3) return

"basilisk", "gnu", kiwi"

a. [3 points]

```java
/** A List consisting of elements I, I+K, I+2K, ... of L,
 * numbering from 0. If I is greater than or equal to the
 * length of L, the result will be the empty list. Requires
 * I>=0, K>0; throws IllegalArgumentException otherwise. Does
 * NOT modify any of the List objects in the original list
```
* pointed to by L. */

static List subseq1(List L, int i, int k)
{
  //FILL THIS IN (about 8 lines)

}

b. [3 points] For this version, do not use the new operator or in any other way allocate new List objects.

static List subseq2(List L, int i, int k)
/** A List consisting of elements I, I+K, I+2K, ... of L,
   * numbering from 0. If I is greater than or equal ot the
   * length of L, the result will be the empty list. Requires
   * I>=0, K>0; throws IllegalArgumentException otherwise. May
   * modify Llist objects in the list pointed to by L. */
{
  //FILL THIS IN (about 15 lines)

}
Problem #2

[1 point] Where does one find a fenestra ovalis?

Problem #3

3. [7 points] The class `Map` is intended to represent functions from `Objects` to `Objects`. The idea is that if `F` is a `Map`, then `F.val(x)` may be thought of as "F of x," and `F.set(x, y)` changes F just enough so that `F.val(x) == y`. Fill in the missing parts below to agree with the comments. You may assume that all arguments are correct, and no error checking is needed.

```java
import java.util.Enumeration;

/** A mapping (function) from Objects to Objects */
class Map {

    // FILL THIS IN WITH THE NECESSARY FIELDS.

    /** A new empty function: initially, val(x) == null for all * x. */
    public Map() {

        // FILL THIS IN

    }

    /** The current value of THIS at X. */
    public Object val(Object x) {

        // FILL THIS IN

    }

    /** A new empty function: initially, val(x) == null for all * x. */
    public Map() {

        // FILL THIS IN

    }

    /** The current value of THIS at X. */
    public Object val(Object x) {

        // FILL THIS IN

    }

```
/** Change THIS so that val(X) == Y, and val(z) is unchanged for all z != X. */

public void set(Object x, Object y) {
    // FILL THIS IN (about 6 lines)
}

/** An Enumeration that yields all Objects, x, such that val(x) == Y. Requires that Y != null. */

public Enumeration inverse(Object y) {
    return __________________________;
}

// FILL THIS IN WITH ANY ADDITIONAL METHODS AND CLASSES YOU NEED.

// FILL THIS IN WITH ANY ADDITIONAL METHODS AND CLASSES YOU NEED.

Problem #4

[6 points] The definition that follow are supposed to be a particularly convoluted way of computing the total number of characters in an array of strings. You are to fill in the blanks and other space provided so as to complete the definition, subject to some constraints. First you may not write any conditional statements: that is, there should be no uses of if, for, do, while, or switch in your program. Instead, use Java's inheritance and dynamic method-calling features. Second, the definitions on this page must make no mention whatever of anything defined on the next page. HINTS: (a) The function DoList.doAll will be tail-recursive; (b) as you can see form the definition of DoList.EMPTY, an empty DoList, unlike an ordinary list, is not represented by the value null.

class DoList extends List {

    public DoList(Object head, DoList tail) {
        super(head, tail);
    }
}
/** The empty DoList (NOTE: This is NOT null). */

final static DoList EMPTY =
new ______________________________;

// FILL IN ANY DEFINITIONS OF FUNCTIONS AND CLASSES

// YOU NEED TO HELP IMPLEMENT EMPTY (ABOVE).

/** Make F act on all elements */

public void doAll(Actor f) {
    f.act(head);

    // FILL IN (ABOUT ! LINE)

}
abstract class Actor {
    abstract void act(Object);
};

class LengthAccum __________________

// FILL IN

class Foo {

    /** The total number of characters contained in all
     * the Strings in array A. */

    public static int totalLength(String[] A) {
        DoList L = DoList.EMPTY;
        for (int i = A.length-1; i >= 0; i-= 1)
            L = new DoList (A[i], L);

Problem #5

[6 points] Fill in the following to agree with the comment. Write the function assuming that the parameters will be correct, and no error checking will be required.

/** Assuming L is a string of the form "(a0, a1, ..., an-1)", where each of the substrings ai contains no commas or parentheses, k return the string *
* "(a0,a1,...,aK-1,X,aK,...,an-1)"
* Assumes that )<=K<=n. For example,
* insert("foo", "(bar, baz, thud)", 0) --> "(foo, bar, baz, thud)"
* insert("foo", "(bar, baz, thud)", 1) --> "(bar, foo, baz, thud)"
* insert("foo", "(bar, baz, thud)", 3) --> "(bar, baz, thud, foo)"
*/

static String insert(String X, String L, int k) {
    // FILL IN