University of Toronto
**csc148—Introduction to Computer Science, Spring 1999**

# First Term Test
# Horton's and Reiter's sections

---

**Duration:** 50 minutes

**Aids allowed:** None

Make sure that your examination booklet has 6 pages (including this one). Write your answers in the spaces provided. Write legibly.

**Family Name:** ____________________ **Lecturer:** ____________________

**First Name:** ____________________ **Tutor:** ____________________

**Student # :** ____________________

1. ________ / 6

2. ________ / 10

3. ________ / 14

Total ________ / 30

## Question 1

[6 marks in total]

Consider the following new ADT: A **FrequencyCounter** contains a set of zero or more items, plus an integer frequency for each item. The operations one can perform on a FrequencyCounter are:

- clear: Clear out the contents of this FrequencyCounter.
- occurrence( item ): Record the occurrence of the 'item' by adding one to that item's frequency count. If the item hasn't occurred before, add it to the set of items and set its count to one.
- average(): Return the average frequency among all the items in the set, or zero if the set is empty.
- mostFrequent(): Return the most frequently occurring item in the set, or null if the set is empty.

*(a)* [4 marks]
Write a Java interface for this ADT. Assume the items in a FrequencyCounter will be instances of class `Object`. Comments are not necessary.

*(b)* [2 marks; no penalty for wrong answers]
Say that class `C` is in package `P` and has a `protected` member `m`. Suppose another class wants to use `m`. Which of the following kinds of class can? Circle the appropriate answer for each.

| | | |
|---|---|---|
| a class that is a subclass of `C` and is in package `P` | CAN | CANNOT |
| a class that is a subclass of `C` and is in a different package | CAN | CANNOT |
| a class that is not a subclass of `C` and is in package `P` | CAN | CANNOT |
| a class that is not a subclass of `C` and is in a different package | CAN | CANNOT |

## Question 2

[10 marks in total]

Assume that the following class has been defined for creating nodes in a linked data structure.

```
class Node {
   public int data;
   public Node next;
   public Node( int n ) {
        data = n;
        next = null;
   }
}
```

Now suppose we are writing a new class that has the following instance variables for keeping track of a linked list:

```
// The first and last Nodes in the linked list.  If the list has just one Node,
// both must refer to it.  If the list is empty, both must be set to null.
private Node first, last;
```

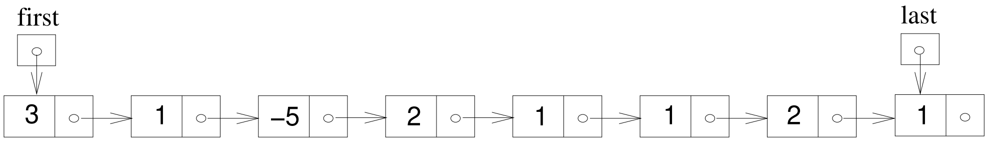
*(a)* [5 marks]
Write the body of the following method, which is to be part of the new class. Hint: Don't forget the case where the linked list is currently empty.

```
// Insert 'newNum' at the end of the linked list.
public void insertAtEnd( int newNum ) {





}
```

*(b)* [3 marks]
The following mysterious method is also to be part of the new class.

```
// Precondition: there is at least one element in the linked list.
public void doSomething( int n ) {
     boolean b = false;
     if (first.data == n) {
            b = true;
     }
     Node current = first.next;
     Node previous = first;
     while (current != null) {
             if (current.data == n && b) {
                 previous.next = current.next;
                 current = current.next;
             }
             else {
                 if (current.data == n) {
                        b = true;
                 }
                 previous = current;
                 current = current.next;
             }
     }
     last = previous;
}
```

Assume that instance variables `first` and `last` have been set to refer to `Node`s in the structure below. Trace what would happen if `doSomething()` were called with parameter value 1.




Show the final state of the data structure here:

*(c)* [2 marks]
Write a precise comment that describes what method `doSomething()` does.

## Question 3

[14 marks in total]

Consider following new interface:

```
import java.util.Enumeration;
public interface Shakable {

   // Insert the given 'items' into this Shakable.  Anything that may have been
   // in the Shakable before is lost.  If all of the items fit into the Shakable
   // return true; if not, stuff as many as possible and return false.
   public boolean stuff( Enumeration items );

   // Randomly reorder the items in the Shakable.
   public void shake();

   // Return an Enumeration of the items currently in the Shakable.
   public Enumeration unstuff();
}
```

Below is an outline for a class that implements `Shakable`. It will store the items to be shaken in a simple array of `Objects`. Declare any instance variables required and complete the constructor below. Then on the next page, complete method `stuff()`.

```
import java.util.Enumeration;
public class Shaker implements Shakable {

   // Instance variables:








   // Construct a Shaker with room for 'capacity' items.
   public Shaker( int capacity ) {








   }

   // The Shaker class is continued on the next page ...
```

```
        // Insert the given 'items' into this Shakable.  Anything that may have been
        // in the Shakable before is lost.  If all of the items fit into the Shakable
        // return true; if not, stuff as many as possible and return false.
        public boolean stuff( Enumeration items ) {

        }

        // Randomly reorder the items in the Shakable.
        public void shake() {  // Details omitted.  }

        // Return an Enumeration of the items currently in the Shakable.
        public Enumeration unstuff() {  // Details omitted.  }
    }
}
```

**END OF TEST**