Mid Term Test – Section L0101

Duration: 50 minutes
Aids allowed: none

Make sure that your examination booklet has 8 pages (including this one). Write your answers in the spaces provided. Write legibly. You may use page 8 for rough work (tear it off, if you like). If you require more space to answer a question, write on the back of the previous page, and indicate in the answer space where you answer is.

Surname:

Given name(s):

Student #:

TA's name: (please circle below)

J. Listgarten   A. Mok   E. Garderman

Please note that if you write the mid term in pencil, you will ***not*** be allowed to submit a remark request.

0.	/ 2
1.	/ 6
2.	/ 10
3.	/ 12
4.	/ 10
Total:	/ 40

Question 0: Administrative

[2 marks]

Write your surname and student number at the top of every page (except page 1) of this test.

Question 1: Design by Contract

 [3 x 2 marks each – 6 marks total]

Below are methods with their comments. There is something wrong with each comment.
Briefly explain what's wrong with each one (if there is more than one problem, pick the one
you feel is most important).

```
/* Finds v in the Doubly-Linked List with head h.
 * Pre: h points to a valid Doubly-Linked list which
 *      contains v. */
static boolean Search (DLLNode h, Object v);
```

Answer:

Does not explain what the method returns.

```
/* Finds the middle item in a Vector
 * Pre: d.size() > 0 */
static Object middle(Vector d);
```

Answer:

Does not explain which vector it is finding the middle of. Does not explain what the
method returns.

```
/* Uses recursion to print the entire BST rooted at r.
 * Pre: r is the root of a valid BST. */
static void printTree(BSTNode r);
```

Answer:

The fact that the method uses recursion is a private internal detail.

Question 2: Interface Implementation & Exceptions

[10 marks]

Below is the code for CircularQueue from the slides, slightly altered (changes are bolded for convenience). Your task is to complete an Iterator class that will be returned by the iterator() method which has been added to the CircularQueue class.

```
public class CircularQueue implements Queue {

    public int size; // The number of elements in me

    // The elements in me, stored in contents[head..tail-1], with wraparound
    public Object[] contents;

    public int head, tail=0;

    /* Representation Invariant:
     * If size is 0, I am empty, and head=tail; otherwise:
     *
     *     Let the capacity be contents.length.
     *     contents[head] is the head.
     *     contents[tail-1] is the tail.
     *     if head < tail,
     *         contents[head .. tail-1] contains the Objects in the order
     *         they were inserted, and size=tail-head.
     *     if head >= tail,
     *         contents[head .. capacity-1, 0 .. tail-1] contains the
     *         Objects in the order they were inserted, and
     *         size=tail-head+capacity. */

    // Append o to me.
    public void enqueue(Object o) {...}

    // Remove and return my front Object.
    // Precondition: I am not empty
    public Object dequeue() {...}

    // Return my front Object.
    // Precondition: I am not empty.
    public Object head() {...}

    // Return the number of Objects in me.
    public Object size() {...}

    // Return an Iterator representation of this Queue.
    public iterator() { return new CQterator(this); }
}
```

You may assume that you have a class called *NoSuchElementException* at your disposal.
Code:

```
class NoSuchElementException extends Exception {}
```

Note that an Iterator does not make a copy of any data, or change the underlying CircularQueue in any way.

Note also that this is not a test of how well you know the Iterator interface. Some features of the interface have been left out. Just complete the methods that have method headers below.

```java
class CQIterator implements Iterator {

   CircularQueue q;
   int numLeft=0;
   int next;

   // q is the CircularQueue for which this CQIterator shall
   // serve as Iterator
   CQIterator(CircularQueue q) {

      next = q.contents[q.head];
      numLeft = q.size();
      this.q = q;


   }// CQIterator()

   // Returns true if there are elements which have not yet
   // been returned.
   public boolean hasNext() {

      return numLeft > 0;

   }// hasNext()


   // Returns the next object in the underlying class.  If there is
   // no next element, then a NoSuchElementException is thrown.
   public Object next() throws NoSuchElementException {

      if (numLeft == 0) {
         // Assertion: there is no next element
         throw new NoSuchElementException();
      }
      // Assertion: numLeft > 0

      int next = this.next++;

      if (this.next == q.contents.length) {
         // Assertion: the next object in the queue is at
         // the beginning of the array (we're wrapping around).
         this.next = 0;
      }

      numLeft--;
      return q.contents[next];

   }// next()
} //CQIterator
```

Question 3: Linked Lists

[12 marks total]

This is the Linked List node class that you will be using for the rest of this question.

```
class LLNode {
   String data;
   LLNode next;
   LLNode prev;

   LLNode(String d, LLNode n, LLNode p) {
      data = d; next = n; prev = p;
   }
}
```

(a) Tracing

[4 marks]

Below is the method foo. Draw a diagram (using boxes & arrows, not the memory model) of
what this linked list will look like when the method is run up to the comment. You may use
the last page of the exam booklet for scrap work – draw only the final diagram below.

```
void foo() {
   LLNode a = new LLNode("fun!",null,null);
   LLNode b = new LLNode("this",a,a);
   LLNode c = new LLNode("such",b,a);
   a.next = b;
   c.prev = new LLNode("is",c,a.next);
   c.next = b.next;
   b.next = c.prev;
   c.prev.prev.prev.prev = b.next.next;
   // DRAW HERE
   System.out.println("That wasn't easy...");
}
```

Drawing:

The drawing is of a doubly-linked list with nodes (in order): "this", "is", "such", "fun". Note that fun.next points to "this" and this.prev points to "fun".

(b) Linked List Operations

[8 marks]

You are to write a method that will move the smallest value in a linked list to the head of the list. The method header is below – complete the code.

Helpful reminder: Strings contain a method called "compareTo" which is used to compare two Strings. Specifically:

compareTo(Object o) returns:
The value 0 if the argument is a string equal to this string; a value less than 0 if the argument is a string greater than this string; and a value greater than 0 if the argument is a string less than this string.

```
/* Finds the smallest value in the Linked List rooted at head
and moves it to the front of the list.  If more than one
String in the list has an equal, lowest value, the one closest
to the head is moved. */

static void SortOne(LLNode head) {
   if (head == null) {
      // if there's no list, there's nothing to sort.
      return;
   }

   LLNode smallest = head;
   LLNode current = head;

   // find the smallest node
   while (current.next != null) {
      if (smallest.data.compareTo(current.data) > 0) {
         smallest = current;
      }
   }

   // If the head is the smallest item, it's already sorted
   if (smallest == head) {
      return;
   }

   // Insert a new node to hold the old head
   head.next.prev = new LLNode(head.item,head.next,head);
   head.data = smallest.data;
   head.next = head.next.prev;

   // Remove the old node which held the smallest
   smallest.prev.next = smallest.next;
   if (smallest.next != null) {
      smallest.next.prev = smallest.prev;
   }

}
```

Question 4: Trees

[10 marks total]

Answer each question in the space provided.

a. You have a tree made up of *n* nodes. What is the maximum height of this tree?
[1 mark]

$$n$$

b. You have a reasonable search algorithm for finding the smallest value in a binary tree. If you have a binary tree with height *h* that is made up of *n* nodes, what is the smallest number of nodes that the algorithm will have to visit in order to be sure it has found the smallest value?
[1 mark]

$$n$$

c. You have learned that singly linked lists are trees. Are circularly linked lists also trees? Why or why not?
[3 marks]

No – one of the properties of a tree is that the root have no parent – in the case of a circularly linked list, the root (the head) has a parent.

d. Below is the code for method "mystery". Write a complete (external) method comment for "mystery".

This is the BSTNode class mystery uses:

```
/* A node for a Binary Search Tree */
class BSTNode { int data; BSTNode right; BSTNode left; }
```

This is the mystery method:

```
static int mystery(BSTNode r) {
   BSTNode temp = r;
   while (temp.left!=null) {
      temp=temp.left;
   }
   return temp.data;
}
```

External comment for this method:
[5 marks]
// Finds and returns the smallest value stored in the BST rooted at r.
// pre: r != null

You may use this page for your rough work.