

CSC 148H L5101 Midterm Fall 2003  
Duration — 50 minutes  
Aids allowed: none

Student Number:

Family Name:  Given Name:

---

*Do not turn this page until you have received the signal to start.*  
(Please fill out the identification section above,  
and read the instructions below.) *Good Luck!*

---

This midterm consists of 3 questions on 5 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.* Comments are not required, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.

# 1: \_\_\_\_\_/12

# 2: \_\_\_\_\_/ 8

# 3: \_\_\_\_\_/10

Write your student number at the bottom of pages 2-5 of this test.

If you use any space for rough work, please indicate clearly what you want marked.

TOTAL: \_\_\_\_\_/30

---

**Question 1.** [12 MARKS]

Suppose we have a class SimpleList:

```
public class SimpleList {

    /** A SimpleList that can hold up to 'capacity' elements. */
    public SimpleList(int capacity) { /* body not shown */ }

    /** Return the number of elements in me. */
    public int size() { /* body not shown */ }

    /** Add 'o' to the end of me. */
    public void add(Object o) { /* body not shown */ }

    /** Insert 'o' at index 'i', moving elements to make room.
        Requires: 0 <= i <= size() < the capacity. */
    public void add(int i, Object o) { /* body not shown */ }

    /** Remove the first occurrence of 'o' from me, if there is one. */
    public void remove(Object o) { /* body not shown */ }

    /** Remove and return the element at index 'i', moving elements to fill the gap.
        Requires: 0 <= i < size(). */
    public Object remove(int i) { /* body not shown */ }

}
```

Write the bodies of the methods in class SillyQueue:

```
/** Uses two SimpleLists capacity 25 to implement a Queue of capacity 50. */
public class SillyQueue implements Queue {

    private SimpleList s1 = new SimpleList(25);
    private SimpleList s2 = new SimpleList(25);

    // Representation invariant:
    //   The head is always s1's 0th element.
    //   If there are at most 25 elements,
    //     s1 contains them in order, and s2 is empty.
    //   If there are more than 25 elements,
    //     s1 contains the first 25, and s2 contains the rest

    public int size() {

        return s1.size() + s2.size(); // 2 marks

    }

}
```

```
public void enqueue(Object o) {

    if (s1.size() < 25) { // 2 marks for logic
        s1.add(o);        // 1 mark for adding
    } else {
        s2.add(o);
    }

}

public Object head() {

    Object result = s1.remove(0); // 2 marks for returning the head.
    s1.add(0, result); // 1 mark for not really removing it.
    return result;

}

public Object dequeue() {

    Object result = s1.remove(0); // 1 mark for removing and returning head
    if (s2.size() > 0) { // 1 mark for detecting when to transfer an element
        s1.add(s2.remove(0)); // 2 marks for transferring element from s2 to s1.
    }
    return result;

}
}
```

**Question 2.** [8 MARKS]

Consider the following class for nodes of a linked list:

```
public class Node {
    public int value;
    public Node next;
}
```

Write the following method:

```
/**
```

```
Return the linked list starting from the first node whose value is 'start'
through to the last node whose value is 'end'.
```

```
For example, if the list contains the elements 3, 7, 2, 7, -1, 2, 8 then
subList(list, 7, 2) returns the linked list consisting of all the *nodes*
of list except the first one and last one.
```

```
Note: THIS WILL USUALLY AFFECT THE ORIGINAL LIST.
```

```
Requires: list refers to a linked list that contains start and end, and
the first node containing start doesn't occur after all the nodes containing end.
```

```
*/
```

```
public static Node subList(Node list, int start, int end) {
```

```
    while (list.value != start) { // 2 marks
        list = list.next;
    }
```

```
    Node current = list;
    Node tail = null;
    while (current != null) { // 1 mark
        if (current.value == end) { // 2 marks
            tail = current;
        }
        current = current.next; // 1 mark
    }
```

```
    tail.next = null; // 1 mark
    return list; // 1 mark
```

```
}
```

**Question 3.** [10 MARKS]

```

public class C {
    public static int i;
    public void p(int j) {
        m(j);
    }
    public void m(int j) {
        i = i + j;
        s();
    }
    public static void s() {
        i = i + 1000;
    }
}

public class D extends C {
    public int i;

    public void m(int j) {
        i = i + j;
        s();
    }
}

public class M {
    public static void main(String[] a) {
        C c = new C();
        c.p(1);
        D d1 = new D();
        d1.m(10);
        C d2 = new D();
        d2.p(100);
    }
}
    
```

Draw the memory model for this program just after the 3rd time that `s()` returns (but before the method that calls `s()` returns).

