University of Toronto

Department of Computer Science

CSC 148H1S — Winter 2004
Section L5101

Midterm test

No aids allowed.

Time: 50 minutes

ANSWERS

1	
2	
3	
Total	

1. [10 marks]

Here is a Java program. It compiles and runs without errors.

```
class A {
    private int ax;
    public A(int x) { ax = x; }
    public void m() {
        System.out.println("A: " + ax);
    }
}

class B extends A {
    private A localA = new A(6);
    private int bx;
    public B(int x) {
        super(x + 7);
        bx = x;
        localA.m();
    }
    public void m() {
        super.m();
        localA.m();
        System.out.println("B: " + bx);
    }
}

public class Supers {
    public static void main(String[] args) {
        A a = new A(22);
        B b = new B(32);
        A ab = new B(42);
        System.out.println("made");
        a.m();
        b.m();
        ab.m();
    }
}
```

Give the output from this program in the space below.

ANSWER:

```
A: 6                        B: 32
A: 6                        A: 49
made                        A: 6
A: 22                       B: 42
A: 39
A: 6
```

2. [10 marks]

The class LList, part of which is provided below, implements lists as ordinary singly-linked lists. We want to allow it to construct and return an Iterator that gives provides the list elements in reverse order.

Complete the ReverseIterator class (an inner class of Llist), begun below. You are not allowed to change anything except the contents of ReverseIterator itself, and ReverseIterator may not change the data stored in the LList.

```
class LList {

    private class Node {
        Object data;
        Node next;
    }

    private Node head; // points to first element of this LList
    // "head" is null if the list is empty. There is no "tail" variable.

    /** Return the number of items in this LList. */
    public int size() { /* implementation irrelevant */ }

    /** Return an ordinary iterator. (Implementation omitted.) */
    public Iterator iterator() { /* implementation irrelevant */ }

    /** Return an iterator that presents the elements of this LList
     * in reverse order. */
    public Iterator reverseIterator() {
        return new ReverseIterator();
    }

    // All other parts of LList are irrelevant and are omitted.

    private class ReverseIterator implements Iterator {
        // Complete this class. You may omit the remove() method, so
        // you need to write only a constructor and two methods:
        // boolean hasNext() -- returns true if there are elements left
        //    to return
        // Object next() -- returns the next element in sequence

        // Reminder: ReverseIterator is an inner class and therefore can
        // refer to private variables of the parent class.

        // You may call iterator(), and you may set up your own
        // data structure, using the standard Java API if you wish.
        // You may not change the LList itself.
```

ANSWER:

```
        private List mylist = new ArrayList();
        private Iterator myit;
        public ReverseIterator() {
            for (Iterator it = iterator(); it.hasNext(); )
                mylist.add(0, it.next());
            myit = mylist.iterator();
        }
        public boolean hasNext() { return myit.hasNext(); }
        public Object next() { return myit.next(); }
    }
```

Comments: I was really thinking of having the students create their own array, copy the object references from the LList into it, and run their own index through it. But on reflection, I'd do it with the API. Here's the original way, which is of course fine:

```
        private Object[] mydata = new Object[size()];
        private int myindex = size() - 1;
```

```
        public ReverseIterator() {
            int where = 0;
            for (Iterator it = iterator(); it.hasNext(); )
                mydata[where++] = it.next();
        }
        public boolean hasNext() { return myindex >= 0; }
        public Object next() { return mydata[myindex--]; }
    }
```

And here's another very nice way, based directly on linked lists:

```
        private Node revHead;

        public ReverseIterator() {
            revHead = null;
            for (Node cur = head; cur != null; cur = cur.next) {
                Node temp = new Node();
                temp.data = cur.data;
                temp.next = revHead;
                revHead = temp;
            }
        }

        public boolean hasNext() { return revHead != null; }

        public Object next() {
            if (! hasNext())
                throw new NoSuchElementException();
            Node temp = revHead;
            revHead = revHead.next;
            return temp;
        }
```

3. [10 Marks]

Let's define an "extrusion" as a three-dimensional object that might be squeezed out of a toothpaste tube. It has a long axis and a cross-section that is the same everywhere along the axis. The cross-section is perpendicular to the axis, and the ends are flat. A cylinder is an extrusion with a circular cross-section, and an ordinary plank of wood is an extrusion with a rectangular cross-section. (In mathematics, the plank would be a "right parallelepiped" or something like that.) For all extrusions, the volume is length × area — that is, the product of the length of the axis times the area of the cross-section.

Here is an abstract class representing extrusions:

```
abstract class Extrusion {
    private double length;
    public double getLength() { return length; }
    public Extrusion(double length) { this.length = length; }
    public abstract double getArea(); // cross-sectional area
    public abstract double getPerimeter();
    public double getVolume() { /* See below. */ }
}
```

(a) Write the class Plank, a subclass of Extrusion. A Plank has width and depth as well as length. Its area is width × depth, and its perimeter is 2 × (width + depth).

Your Plank class must:

- Have a constructor that set the size parameters of the object (width, depth and length).
- Have "get" methods that return the size parameters.
- Be concrete, not abstract.

ANSWER:

```
class Plank extends Extrusion {
    private double width;
    private double depth;
    public Plank(double len, double wid, double dep) {
        super(len);
        width = wid;
        depth = dep;
    }
    public double getArea() { return width*depth; }
    public double getPerimeter() { return 2*(width + depth); }
    public double getWidth() { return width; }
    public double getDepth() { return depth; }
}
```

(b) Write the getVolume() method of the Extrusion class, which is not complete in the version above.

```
public double getVolume() {
    // Complete the rest of this method.
ANSWER:
    return getArea() * length;
}
```