Introduction to Computer Science II (CSI 1101)
Midterm Examination

Instructor: Marcel Turcotte

February 2003, duration: 2 hours

Identification

Student name: ______________________________

Student number: ____________________ Section: ____________________

Instructions

1. This is a closed book examination.
2. No calculators or other aids are permitted.
3. Write comments and assumptions to get partial marks.
4. Beware, poor hand writing can affect grades.
5. Do not remove the staple holding the examination pages together.
6. Write your answers in the space provided. Use the backs of pages if necessary. You may **not** hand in additional pages.

Marking scheme

Question	Maximum	Actual
1	38	
2	18	
3	18	
4	10	
5	16	
Total	**100**	

Question 1 (38 marks)

(a) Write an interface called **Pair**. A *pair* represents a group of two *elements*. The interface must contain the following methods:

- **Object getFirst();**
- **Object getSecond();**
- **void swap();**

(b) Write a linked-list implementation for the *abstract data type* **Pair**. This class, called **LinkedPair**, must implement the interface **Pair** and use the class **Node** below to store its elements:

```
class Node {
    protected Object value;
    protected Node next;
    protected Node(Object value, Node next) {
        this.value = value;
        this.next = next;
    }
}
```

Variables

The class **LinkedPair** must have an instance variable called **first** to designate the first node of the linked-list. No other instance variable is necessary or allowed.

Constructor

Write a constructor with two parameters, both of type **Object**, that are used to initialize the first and second elements of the list.

Methods

Write all the necessary methods so that **LinkedPair** implements the interface **Pair**.

boolean equals(Object obj)

Override the definition of the method **equals(Object obj)** so that the method returns **true** *i)* if **obj** designates an object from a class that also implements the interface **Pair** and *ii)* both data structures contain equivalent objects, in the same order.

Hint: "**o instanceof I**" can be used to determine if the object designated by the reference variable **o** is an object of a class that implements the interface **I**.

Question 1b (continued)

(c) Write an array-based implementation for the *abstract data type* **Pair**. This class, called **ArrayPair**, must implement the interface **Pair** and use an array to store its elements.

Variables

The class **ArrayPair** must have an instance variable called **elements** to designate an array of two elements. No other instance variable is necessary or allowed.

Constructor

Write a constructor with two parameters, both of type **Object**, that are used to initialize the first and second elements of the array.

Methods

Write all the necessary methods so that **ArrayPair** implements the interface **Pair**.

boolean equals(Object obj)

Override the definition of the method **equals(Object obj)** so that the method returns **true** *i)* if **obj** designates an object from a class that also implements the interface **Pair** and *ii)* both data structures contains equivalent objects, in the same order.

Question 1c (continued)

Question 1 (continued)

The implementation of the classes written in parts b and c should be such that the following test program would print “bravo”.

```
public class Test {
    public static void main(String[] args) {

        Pair a = new LinkedPair(new String("data"), new String("structure"));
        Pair b = new ArrayPair(new String("structure"), new String("data"));

        if (a.equals(b)) {
            System.out.println("wrong :-(");
        } else {
            a.swap();
            if (a.equals(b)) {
                System.out.println("bravo !");
            } else {
                System.out.println("wrong :-(");
            }
        }
    }
}
```

Question 2 (18 marks)

(a) Create a class hierarchy to represent vehicles:

- All vehicles have a weight (double);
- A car is a specialized vehicle that also has a number of passengers (int);
- A truck is a specialized vehicle that carries a certain load (double);
- A truck can load and unload its payload. Trying to unload more weight than the total weight of the payload has no effect (the method must also return false);
- The weight of a truck is the weight of the vehicle plus its payload.
- All the weights are expressed in tonnes.

Question 2 (continued)

(b) Complete the definition of the method **calculateFees** for the class **Ferry** below.
The method **calculateFees** returns the total of all the fees for each vehicle in the ferry.

- the passage fee for a car is \$ 10 per tonne plus \$ 5 per passenger;
- the passage fee for a truck is \$ 100 per tonne.

```
class Ferry {
    private Vehicle[] vehicles;

    Ferry(Vehicle[] vehicles) {
        this.vehicles = vehicles;
    }

    double calculateFees() {
        // complete the method

    }
}
```

Hint: "**o instanceof I**" can be used to determine if the object designated by the reference variable **o** is an object of a class that implements the interface **I**.

Question 3 (18 marks)

In the class **SimpleList** below, complete the code for the method **remove(Object obj)** so that it removes the left most occurrence of **obj** in the list. For example, for the list a, b, c, b, d removing b would change the list to a, c, b, d.

```
public class SimpleList {

    private static class Node {
        private Object value;
        private Node next;
        Node(Object value, Node next) {
            this.value = value;
            this.next = next;
        }
    }

    private Node first;

    public void remove(Object obj) {

        // special case: this list is emtpy
        if (                         )
            return;

        // special case: obj is the first element
        if (                         ) {




        } else {
            // general case:














        }
    }
```

Question 4 (10 marks)

In the class **ArrayList** below, implement the method **toArray()**. The class **ArrayList** implements a variable size array so that an unlimited number of elements can be stored. The method **toArray()** returns an array that *i)* has the same size as the number of elements currently stored in the data-structure and *ii)* contains all the same elements, in the same order.

```
public class ArrayList {

    private Object[] elements;
    private int last;

    public ArrayList(int capacity) {
        elements = new Object[capacity];
        last = 0;
    }

    public int size() {
        return last;
    }

    private void increaseSize() {

        Object[] tmp = elements;
        elements = new Object[tmp.length * 3 / 2];

        for (int i=0; i < last; i++)
            elements[i] = tmp[i];
    }

    public void add(Object t) {
        if (last == elements.length)
            increaseSize();

        elements[last] = t;
        last++;
    }
    // complete the method toArray
    Object[] toArray() {










    }
} // end of ArrayList
```

Question 5 (16 marks)

Given the following class and interface definitions:

```
public interface I {
    public void i();
}

public abstract class A implements I {
    private int a = 1;
    public int get () { return a; }
}

public class B  extends A {
    private int b = 2;
    public int get () { return b; }
    public int i() { return a + b; }
}
```

(a) For each of the following, indicate if the statements are **valid** or **invalid**.

i)
```
I o = new B();
o.i();
```

ii)
```
I o = new I();
```

iii)
```
I o = new B();
o.get();
```

iv)
```
B o = new B();
```

(b) **True** or **false** questions.

i) The value 2 will be printer. **true** or **false**

```
B o = null;
System.out.println(o.get());
```

ii) The value 2 will be printer. **true** or **false**

```
A o = new B();
System.out.println(o.get());
```

iii) The value 1 will be printer. **true** or **false**

```
A o = new A();
System.out.println(o.get());
```

(c) Given the following method definition:

```
public static void mystery(String s) {
    s = "NEW";
}
```

What should be the output produced by the following statements:

```
String x = "OLD";
mystery(x);
System.out.println(x);
```

(blank space)