Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

Introduction to Computer Science II (CSI 1101)
Midterm Examination

Instructor: Marcel Turcotte

February 2005, duration: 2 hours

Identification

Student name: _____

Student number: _____ Signature: _____

Instructions

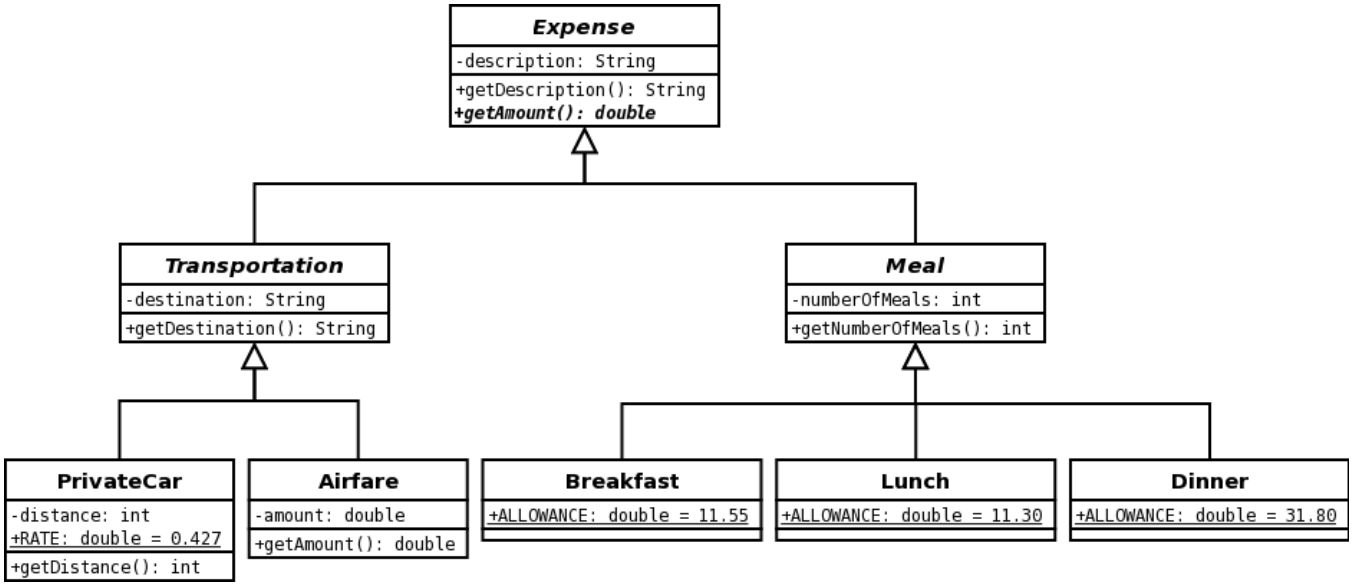
1. This is a closed book examination;
2. No calculators or other aids are permitted;
3. Write comments and assumptions to get partial marks;
4. Beware, poor hand writing can affect grades;
5. Do not remove the staple holding the examination pages together;
6. Write your answers in the space provided.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

Marking scheme

Question	Maximum	Result
1	35	
2	20	
3	15	
4	20	
5	10	
Total	**100**	

Question 1: Inheritance (35 marks)

The company Java Financial Solutions is developing a new software system for tracking professional expenses. You are part of the software development team responsible for the hierarchy of classes to represent expenses.



Specifications:

- All expenses have a description (a character string);
- All the transportation expenses have a destination (a character string);
- A transportation expense using a private car has a distance (of type int);
- A transportation expense by air has a fixed amount (of type double) specified when a new transportation expense is created;
- All the meal expenses have an attribute which represents the number of meals.
- All the expenses have a method to calculate the amount represented by this expense:
  - The amount for a transportation expense using a private car is a fixed rate times the distance traveled;
  - The amount for a transportation expense by air is a fixed amount (specified when a new transportation expense is created);
  - The amount for a meal expense is the number of meals times a fixed rate. The rate depends on the kind of meal: Breakfast, Lunch or Dinner;

(Question 1: continued)

You must write the Java implementation of the following classes. Make sure to include the constructors, the access methods and where appropriate the method for calculating the amount represented by the expense.

A. Expense

B. Transportation

C. PrivateCar

D. Airfare

E. Meal

F. Breakfast

(Question 1: continued)

G. Complete the partial implementation of the class **ExpenseTracker** below. An ExpenseTracker is used to store Expenses. i) Add the type of the elements of the array expenses. ii) Complete the constructor. iii) Complete the implementation of the method **double getTotal()**. The method **double getTotal()** returns the total amount for all the expenses that are currently stored in the ExpenseTracker.

```
public class ExpenseTracker {

    private __________[] expenses;
    private int size; // keeps track of the number of elements

    public ExpenseTracker( int capacity ) {
        ___________________________________;
        size = 0;
    }
    // a method has been defined for adding expenses to the tracker
    public boolean add( Expense e ) { ... }

    public double getTotal() {









    }
}
```

Below is a test program to help you understand the work that needs to be done for this question.

```
public class Run {
    public static void main( String[] args ) {
        ExpenseTracker epro = new ExpenseTracker( 10 );
        epro.add( new PrivateCar( "ACFAS 2004", "Montreal (QC)", 430 ) );
        epro.add( new Airfare( "IWBRA 2005", "Atlanta (GA)", 204.0 ) );
        epro.add( new Breakfast( "IWBRA 2005", 2 ) );
        epro.add( new Lunch( "IWBRA 2005", 3 ) );
        epro.add( new Dinner( "IWBRA 2005", 2 ) );
        System.out.println( "total = " + epro.getTotal() );
    }
}
```

Its output is as follows.

```
total = 508.21000000000004
```

Question 2: Using a stack (20 marks)

In assignment 2, a notation was introduced to represent interactions (**pairs**) between nucleotides (**nucs**) in an RNA molecule. For this question, you must write a method that counts the total number of mismatch pairs. A **mismatch pair** is an interaction that is **not** a Watson-Crick pair (A:U,C:G,G:C or U:A) nor a Wobble (G:U or U:G).

I am reproducing the description of the notation from assignment 2 below. It consists of opened and closed parentheses, as well as the full stop. Each element of the notation is associated with an element of the nucleotides sequence; it is associated with the element that is found at the same location. The notation therefore specifies interactions between nucleotides. The symbol "(" denotes the first element of a pair, the corresponding matching ")" represents the other element of the pair. The symbol "." indicates that the nucleotide at the corresponding position in the sequence is not involved in a pair. Here is a simple example. The first line is not part of the input, it simply shows the positions along the sequence and the interactions string.

```
012345678
AGCUUCGAU
(((...)))
```

In the example above, there are two canonical pairs, 0:8 (A:U) and 2:6 (C:G), as well as one mismatch, 1:7 (G:A). The total number of mismatches is therefore one.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7
012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
GCGAACGGGGCUGGCUUGGUAAUGGUACUCCCCUGUCACGGGUGAGAAUGUGGGUUCAAAUGCCAUCGGUCGCGCCA
(((((((..((((.........)))).(((((.......))))).....(((((.......)))))))))))....
```

In the example above, there are three mismatches, 4:68 (A:G), 28:42 (U:U), 53:61 (G:G). The total number of mismatches is 3.

- Write the method **countMismatches(String nucs, String pairs)** for the class **Rna** below;
- The algorithm must make use of a stack and follow the same general strategy as for the assignment 2;
- You can assume the existence of a class called **LinkedStack** that implements the interface **Stack**.

(Question 2: continued)

```
public class Rna {

  public static int countMismatches( String nucs, String pairs ) {

  } // end of countMismatches
} // end of Rna
```

Question 3: LinkedStack implementation (15 marks)

In the class **LinkedStack** below, implement the method **Object[] toArray()**. **LinkedStack** implements the interface **Stack** using linked elements so that an unlimited number of elements can be stored.

The method **Object[] toArray()** returns an array *i)* of the same size as the number of elements that are currently stored in the linked stack and *ii)* containing all the same elements. The **bottom** element must be stored at position 0, the second element from the bottom must be stored at position 1, etc. so that the top element is stored at position **size()-1**. The stack must remain unchanged.

Here is a test program to help you understand the implementation of the method **Object[] toArray()**.

```
public class Test {
    public static void main( String[] args ) {
        LinkedStack s = new LinkedStack();

        s.push( "A" );
        s.push( "B" );
        s.push( "C" );
        s.push( "D" );

        Object[] array = s.toArray();

        for ( int i=0; i<array.length; i++ ) {
            System.out.println( "array[ " + i + " ] = " + array[ i ] );
        }
    }
}
```

Its output is as follows.

```
array[ 0 ] = A
array[ 1 ] = B
array[ 2 ] = C
array[ 3 ] = D
```

- **LinkedStack** has a method called **int size()** that returns the number of elements that are currently stored in this stack;
- **LinkedStack** implements all the methods of the interface **Stack**, however, **none of them** can be used to implement the method **Object[] toArray()**, you must *traverse* the stack and copy the elements one by one;
- The method **Object[] toArray()** does not change the state of the stack (i.e after a call to the method toArray, the stack must contain the same elements, in the same order, as before the call);
- You are allowed to use the method **int size()** to implement the method **Object[] toArray()**.

(Question 3: continued)

```
public class LinkedStack implements Stack {

    private static class Elem { // Nodes of the linked structure
        private Object value;
        private Elem next;
        private Elem( Object value, Elem next ) {
            this.value = value;
            this.next = next;
        }
    }

    private Elem topElem; // Instance variables
    private int size;

    public int size() { return size; }

    // The other methods would be here.

    public Object[] toArray() {  // Complete the implementation









    } // end of toArray
} // end of LinkedStack
```

Question 4: Generic data structures (20 marks)

Let's define an indexable data structure, called a **Sequence**, to store **Object**s. Like an array, an index (of type int) must be used to access the content of a Sequence. Unlike an array, the lower bound is not necessarily the position 0. The initial bounds are specified when it is created. For instance, "**Sequence ss = new Sequence(100, 200)**" creates a new Sequence to hold 101 objects such that the lowest index is 100 and the highest one is 200. Also unlike the traditional array, a Sequence can grow dynamically. A partial implementation of this class can be found on the next page.

A. (8 marks) Make all the necessary changes to the class Sequence so that exceptions are thrown if the parameters are not valid.

- When creating a new Sequence an exception of type IllegalArgumentException must be thrown if the lower bound is larger that the higher bound;
- An exception of type IndexOutOfBoundsException must be thrown whenever an attempt is made at accessing a position outside of the range of valid indices;
- A null value is considered an illegal argument. An exception of type IllegalArgumentException will occur if a user makes an attempt at adding a null value to a Sequence;
- When increasing the upper bound the new index must be larger than the current highest index;
- IllegalArgumentException and IndexOutOfBoundsException are RuntimeExceptions.

B. (12 marks) Implement the method **increaseHigh(int newIndex)** which increases the capacity of a Sequence. Before a call to the method increaseHigh this Sequence can store **high** - **low** + 1 elements. After a call to the method increaseHigh, this sequence has been changed so that it can store **newIndex** - **low** + 1 elements.

Make all the changes directly into the class Sequence below. Use labels if necessary to indicate the place where a change should be inserted.

```
public class Sequence {

    // instance variables

    private Object[] elems; // used to store the Objects of this Sequence

    private int low;        // the lowest bound
    private int high;       // the highest bount

    public Sequence( int low, int high ) {

        int size = high - low + 1;

        this.low = low;
        this.high = high;
```

```
        elems = new Object[ size ];
    }

    public int getLowIndex() {

        return low;

    }

    public int getHighIndex() {

        return high;

    }

    public void set( int pos, Object obj ) {

        elems[ pos - low ] = obj;

    }

    public Object get( int pos ) {

        return elems[ pos - low ];

    }

    public void increaseHigh( int newIndex ) {

        // complete the implementation









    } // end of increaseHigh
} // end of Sequence
```

Question 5: Short answer questions (10 marks)

A. (4 marks) For this question, we added the methods **findAndReplace** and **main**, shown on the next page, to the class **Sequence** from question 4. Which of the following outputs corresponds to the execution of the method **main**.

(a) Throws IndexOutOfBoundsException: pos=6;

(b) "Count=0, true, 3.141592653589793, true, 3.141592653589793, true, 3.141592653589793";

(c) "Count=2, false, 3.141592653589793, true, 3.141592653589793, false, 3.141592653589793";

(d) "Count=3, false, 3.141592653589793, false, 3.141592653589793, false, 3.141592653589793";

(e) None of the above.

B. (6 marks) Use the stack-based algorithm seen in class to evaluate the following postfix expression and show the content of the stack immediately before and after processing each operator.

$$4 \; 2 \; 5 \; \times \; 8 \; 4 \; / \; - \; \times$$

(Question 5: continued)

```
public class Sequence {

    private Object[] elems;
    private int lo;
    private int hi;

    // The other methods of the class Sequence would be here, see Question 4

    public int findAndReplace( Object src, Object dst ) {
        int count = 0;

        for ( int i=getLowIndex(); i<=getHighIndex(); i++ ) {
            if ( get( i ) == src ) {
                set( i, dst );
                count++;
            }
        }

        return count;
    }

    public static void main( String[] args ) {

        Sequence s = new Sequence( 5, 10 );
        Boolean a = new Boolean( true );

        s.set(  5, a );
        s.set(  6, new Double( Math.PI ) );
        s.set(  7, new Boolean( true ) );
        s.set(  8, new Double( Math.PI ) );
        s.set(  9, a );
        s.set( 10, new Double( Math.PI ) );

        int count = s.findAndReplace( a, new Boolean( false ) );

        System.out.print( "count=" + count + ", " );
        for( int i=5; i<10; i++ ) {
            System.out.print( s.get( i ) + ", " );
        }
        System.out.println( s.get( 10 ));

    }
}
```

(blank space)