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 Computing II (ITI 1121)
Midterm Examination

Instructor: Marcel Turcotte

February 2010, 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 back of pages if necessary. You may **not** hand in additional pages.

Marking scheme

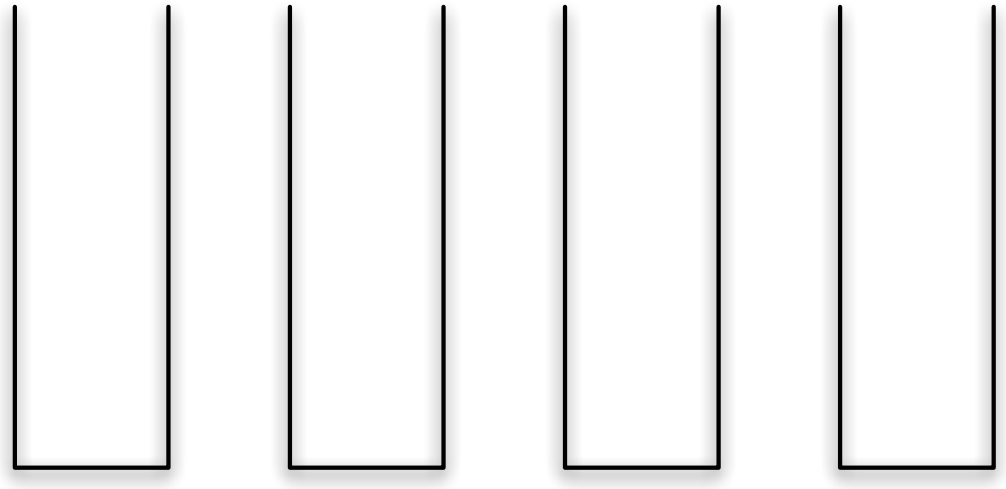
Question	Maximum	Result
1	5	
2	5	
3	5	
4	20	
5	25	
Total	**60**	

Question 1: (5 marks)

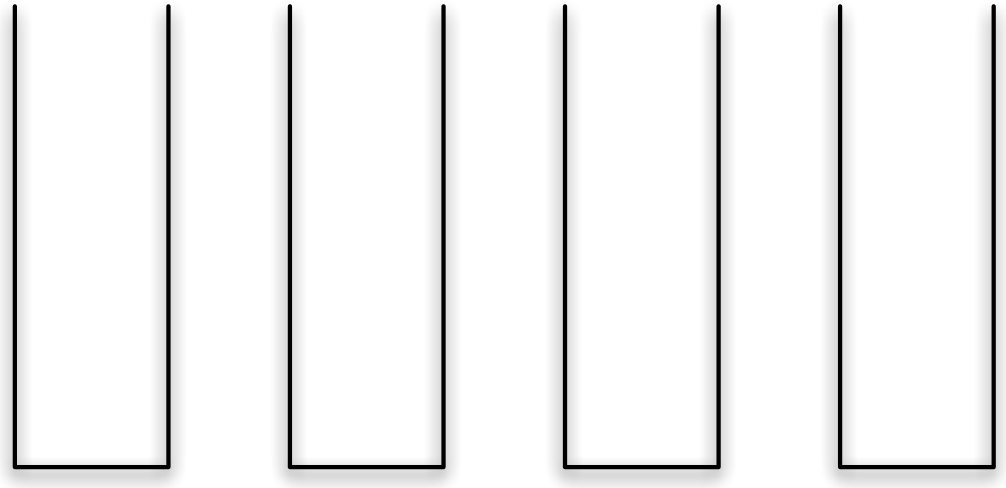
A. Transform the infix expression "$100 - (6 - 2) \times 5$" into postfix (RPN).

B. Use the stack-based algorithm seen in class to evaluate the following postfix (RPN) expression and show the content of the stack immediately **before** and **after** processing each operator. The symbol ␣ represents a space.

$$2 \text{␣} 5 \text{␣} \times \text{␣} 9 \text{␣} 6 \text{␣} 3 \text{␣} \div \text{␣} - \text{␣} +$$



Before $\times$ After $\times$ Before $\div$ After $\div$



Before $-$ After $-$ Before $+$ After $+$

Question 2: (5 marks)

On the next page, you will find the declarations of the classes **Animal**, **Cat** and **FamousCat**. What will be the outcome of compiling and executing each block of code below? Use one of the following four patterns to answer and give a brief explanation.

i. Produces a **compile-time error**;

ii. Produces an **execution-time error**;

iii. Compiles and runs, but **displays nothing**;

iv. Otherwise, write down the information that will be **displayed on the screen**.

A.
```
Animal boo = new Cat( 12.0 );
System.out.println( boo.says() );
```
Answer:

B.
```
FamousCat sylvester = new FamousCat( "That's Not All Folks!" );
System.out.println( sylvester.says() );
```
Answer:

C.
```
Cat garfield = new FamousCat( "It's My Birthday!" );
System.out.println( garfield.says() );
```
Answer:

D.
```
Cat garfield = new FamousCat( "It's My Birthday!" );
FamousCat sylvester = new FamousCat( "That's Not All Folks!" );
sylvester = garfield;
System.out.println( sylvester.says() );
```
Answer:

E.
```
Cat garfield;
Animal boo = new Cat( 12.0 );
FamousCat sylvester = new FamousCat( "That's Not All Folks!" );
if ( boo instanceof Cat ) {
    garfield = (Cat) boo;
    sylvester.swap( garfield );
    System.out.println( sylvester.getWeight() );
}
```
Answer:

```
public abstract class Animal {
    protected double weight = 0.0;

    public Animal( double w ) {
        weight = w;
    }

    public abstract String says();

    public double getWeight() {
        return weight;
    }
}

public class Cat extends Animal {
    public Cat( double w ) {
        super( w );
    }

    public String says() {
        return "Meow!";
    }
}

public class FamousCat extends Cat {
    private String message;

    public FamousCat( String msg ) {
        super( 19.0 );
        message = msg;
    }

    public String says() {
        return message;
    }

    private static void swap( double a, double b ) {
        double tmp = a;
        a = b;
        b = tmp;
    }

    public void swap( Cat other ) {
        swap( weight, other.weight );
    }
}
```

Question 3: (5 marks)

```
public class Pair {
    private Object first;
    private Object second;
    public Object getFirst() {
        return first;
    }
    public void setFirst( Object first ) {
        this.first = first;
    }
    public Object getSecond() {
        return second;
    }
    public void setSecond( Object second ) {
        this.second = second;
    }
    public String toString() {
        return "{" + first + "," + second + "}";
    }
}
```

Given the above declaration of the class **Pair**.

A. Give the result of executing the statements below, i.e. what will be displayed on the screen immediately after the execution of "**System.out.println(p1)**";

B. Draw the memory diagrams representing **p1** and **p2** immediately after executing the last statement (make sure to include all the objects and all the variables involved).

```
Pair p1, p2;
p1 = new Pair();
p1.setFirst( new String( "Alpha" ) );
p2 = new Pair();
p2.setFirst( new String( "Bravo" ) );
p1.setSecond( p2 );
p2.setSecond( null );
System.out.println( p1 );
```

Question 4: (20 marks)

In mathematics, a series is an infinite sequence of terms added together. The **partial sum of the series**, S_n, is the sum of the first n terms.

$$S_n = \sum_{i=1}^{n} a_i$$

Here, you must create a class hierarchy, as illustrated by the UML diagram below, such that all the series have a method **next**, which returns the next S_n. The first call to the method **next** returns S_1, which is

$$a_1$$

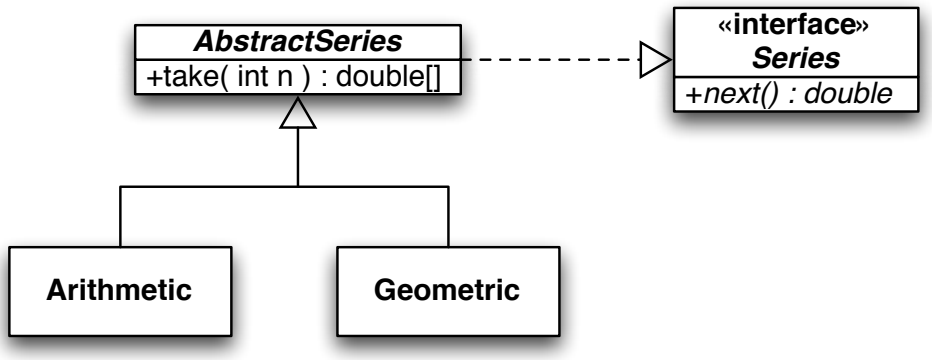
the next call to the method **next** returns S_2, which is

$$a_1 + a_2$$

the next call to the method **next** returns S_3, which is

$$a_1 + a_2 + a_3$$

and so on. The implementation of the method **next** is specific to the type of series, here **Arithmetic** and **Geometric**. Specifically, this hierarchy consists of the interface **Series**, an abstract class called **AbstractSeries**, as well as two concrete implementations, called **Arithmetic** and **Geometric**.



Here is a test program that illustrates the intended use of the classes.

```
AbstractSeries sn;
double[] tuple;
sn = new Arithmetic();
for ( int n=0; n<5; n++ ) {
    System.out.println( sn.next() );
}
sn = new Geometric();
tuple = sn.take( 5 );
for ( int n=0; n<5; n++ ) {
    System.out.println( tuple[ n ] );
}
```

The first loop displays the values 1.0, 3.0, 6.0, 10.0, 15.0, whilst the second one displays, 1.0, 1.5, 1.75, 1.875, 1.9375. See page 10 for the details of the calculations.

(**Question 4: continued**)

A. Create an interface called **Series**. It declares a method called **next** that has a return-type **double**.

B. Write the implementation of the abstract class **AbstractSeries**. It realizes the interface **Series**. The class implements a method called **take** that returns an array containing the next **k** partial sums of this series, where **k** is the formal parameter of the method **take**.

(**Question 4: continued**)

C. Implement the class **Arithmetic**, which is a subclass of **AbstractSeries**. In this class, the first call to the method **next** returns the value 1.0, the second call returns the value 3.0, the third call returns 6.0, the fourth call returns 10.0, etc. The general formula is as follows, the ith call to the method **next** returns $S_{i-1} + i$, where $i \in 1, 2, 3 \ldots$ and S_{i-1} is the value that was returned by the previous call to the method **next**.

$$S_n = \sum_{i=1}^{n} i$$

(**Question 4: continued**)

D. Implement the class **Geometric**, which is a subclass of **AbstractSeries**. Each call to the method **next** produces the next partial sum of the series according to the formula below. The first call returns 1.0, the second call returns 1.5, the third call returns 1.75, etc. You can use **Math.pow(a, b)**, which returns a^b, for your implementation.

$$S_n = \sum_{i=0}^{n} \frac{1}{2^i}$$

(**Question 4: continued**)

The first 5 partial sums of the arithmetic series are

$S_1 = 1$

$S_2 = 1 + 2 = 3$

$S_3 = 1 + 2 + 3 = 6$

$S_4 = 1 + 2 + 3 + 4 = 10$

$S_5 = 1 + 2 + 3 + 4 + 5 = 15$

The first 5 partial sums of the geometric series are

$S_1 = 1$

$S_2 = 1 + \frac{1}{2} = 1.5$

$S_3 = 1 + \frac{1}{2} + \frac{1}{4} = 1.75$

$S_4 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = 1.875$

$S_5 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = 1.9375$

A call to the method **next** produces the next partial sum, i.e. the next value of S_n.

Question 5: (25 marks)

For this question, you must complete the implementation of the class **Tuple**, the interface **Predicate**, and the class **IsNegative**. Here is a test program that illustrates their intended use. The last statement displays the value 2.

```
Tuple<Integer> t;
int n;
n = 5;
t = new Tuple<Integer>( n );
t.add( 3 );
t.add( -14 );
t.add( -15 );
t.add( 9 );
t.add( 26 );
System.out.println( t.count( new IsNegative() ) );
```

A. Complete the implementation of the class **Tuple** on the next page. In computer science, an n-tuple is a linear collection of n elements. The type of the elements is a type argument of the class declaration, in other words, **Tuple** is a parameterized type.

- The implementation uses a fixed size array to store the elements of the collection;
- The class has a constructor that has the following signature **Tuple(int n)**, where **n** is the capacity (physical size) of the tuple;
- The class has a method **boolean add(E elem)**, where **E** is the type of the elements of the collection. If the array is not full, the method adds the element at the end of the collection and returns **true**. Otherwise, if the collection was full when the call was made, the tuple remains unchanged and the method returns **false**;
- Implement the method **int count(Predicate<E> p)**. The method returns the number of elements of this tuple for which the method **eval** of the predicate **p** returns **true**.

(**Question 5: continued**)

```
public class Tuple               {

    // instance variable(s)




    public Tuple( int n ) {









    }

    public boolean add( E elem ) {









    }

    public int count( Predicate<E> p ) {









    }

}
```

(**Question 5: continued**)

B. Complete the implementation of the interface **Predicate**. This interface is a parameterized type. The interface declares a method called **eval** that has a single parameter, its type is the type parameter of the interface. The return-type of the method **eval** is boolean.

```
public                Predicate     {









}
```

C. Complete the implementation of the class **IsNegative**. The class must realize the interface **Predicate**. The method **eval** returns **true** if the value of the parameter is negative, and **false** otherwise.

```
public            IsNegative                                        {









}
```

(blank space)

(blank space)