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 à l'informatique II (ITI 1521)
Examen mi-session

Instructeur: Marcel Turcotte

Février 2010, durée : 2 heures

Identification

Nom, prénom : _____

Numéro d'étudiant : _____ Signature : _____

Consignes

1. Livres fermés ;
2. Sans calculatrice ou toute autre forme d'aide ;
3. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle ;
4. Écrivez lisiblement, votre note en dépend ;
5. Commentez vos réponses ;
6. Ne retirez pas l'agrafe.

Barème

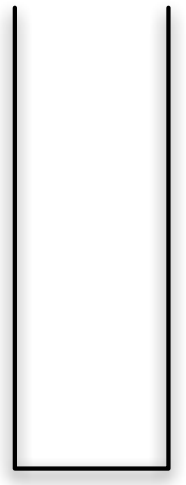
Question	Maximum	Résultat
1	5	
2	5	
3	5	
4	20	
5	25	
Total	**60**	

Question 1 : (5 points)

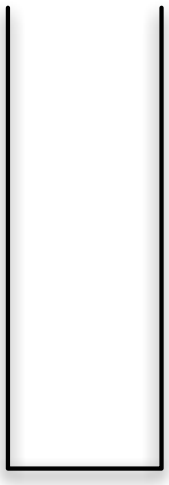
A. Donnez l'expression postfixe (notation polonaise inverse) correspondant à l'expression infixe suivante : $100 - (6 - 2) \times 5$.

B. Utilisez l'algorithme à base de pile vu en classe afin d'évaluer l'expression postfixe (RPN) suivante et montrez le contenu de la pile immédiatement **avant** et **après** le traitement de chaque opérateur. Le symbole ␣ représente une espace.

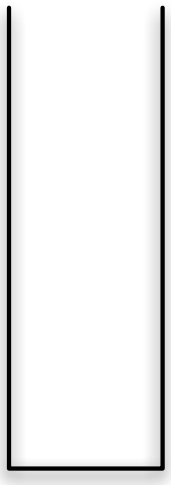
$$2 \text{␣} 5 \text{␣} \times \text{␣} 9 \text{␣} 6 \text{␣} 3 \text{␣} \div \text{␣} - \text{␣} +$$



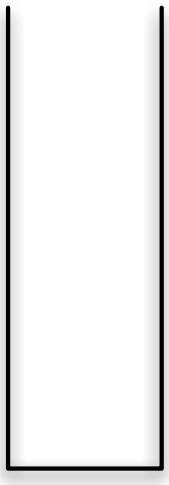
Avant $\times$



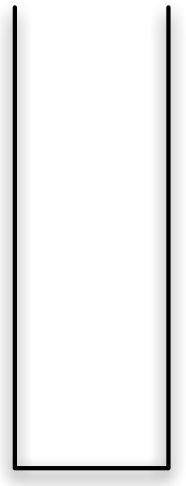
Après $\times$



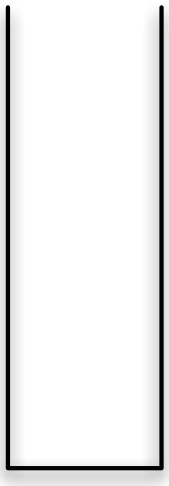
Avant $\div$



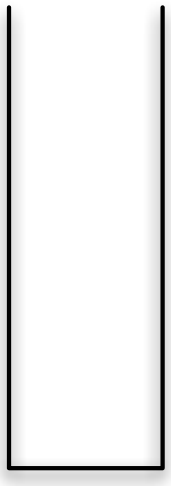
Après $\div$



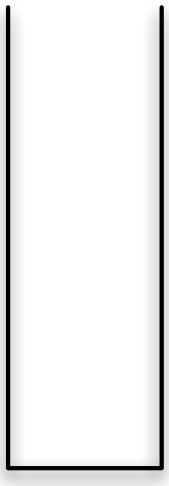
Avant $-$



Après $-$



Avant $+$



Après $+$

Question 2 : (5 points)

Vous trouverez sur la page qui suit les déclarations des classes **Animal**, **Cat** et **FamousCat**. Qu'obtient-on si l'on compile et exécute chaque bloc de code ci-bas ? Utilisez l'une des quatre formes suivantes pour répondre aux questions et donnez une brève explication.

i. Produira une **erreur de compilation** ;

ii. Produira une **erreur d'exécution** ;

iii. Ne produira aucune erreur, mais **n'affichera rien à l'écran** ;

iv. Sinon, donnez l'information qui **sera affichée à l'écran**.

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

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

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

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() );
```
Réponse :

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() );
}
```
Réponse :

```
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 points)

```
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 + "}";
    }
}
```

Étant donné la déclaration de la class **Pair** ci-haut.

A. Donnez le résultat de l'exécution des énoncés ci-bas, c'est-à-dire ce qui sera affiché à l'écran à la suite de l'exécution de l'énoncé « **System.out.println(p1)** » ;

B. Dessinez les diagrammes de mémoire représentant **p1** et **p2** à la suite de l'exécution du dernier énoncé (assurez-vous de dessiner tous les objets et toutes les variables impliqués).

```
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 points)

En mathématique, une série est une séquence infinie de termes qui sont additionnés les uns aux autres. La **somme partielle d'une série**, S_n, est la somme des n premiers termes.

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

Vous devez concevoir une hiérarchie de classes, conforme au diagramme UML ci-bas, de sorte que toutes les séries possèdent une méthode **next**, qui retourne la prochaine valeur S_n. Le premier appel à la méthode **next** retournera S_1, soit

$$a_1$$

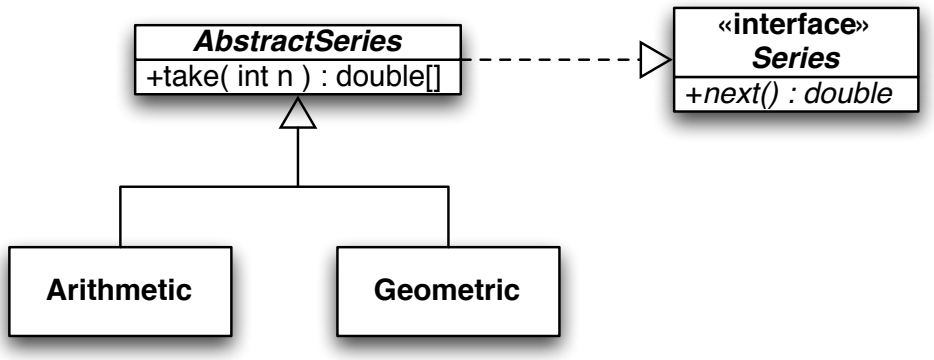
le prochain appel à la méthode **next** retournera S_2, soit

$$a_1 + a_2$$

le prochain appel à la méthode **next** retournera S_3, soit

$$a_1 + a_2 + a_3$$

et ainsi de suite. L'implémentation de la méthode **next** dépend du type de série, **Arithmetic** ou **Geometric** pour cette question. Cette hiérarchie comprend l'interface **Series**, une classe abstraite nommée **AbstractSeries**, ainsi que deux implémentations concrètes, **Arithmetic** et **Geometric**.



Voici un programme test qui illustre l'utilisation de ces 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 ] );
}
```

La première boucle affichera les valeurs 1.0, 3.0, 6.0, 10.0, 15.0, alors que la seconde affichera, 1.0, 1.5, 1.75, 1.875, 1.9375. Le détail des calculs se trouve à la page 10.

(**Question 4: suite**)

A. Vous devez concevoir une interface nommée **Series**. Elle doit déclarer une méthode nommée **next** dont le type de la valeur de retour est **double**.

B. Donnez l'implémentation d'une classe abstraite nommée **AbstractSeries**. Elle doit réaliser l'interface **Series**. La classe implémente la méthode **take** qui retourne un tableau contenant les prochaines **k** sommes partielles de cette série, où **k** est le paramètre formel de la méthode **take**.

(**Question 4: suite**)

C. Donnez l'implémentation de la classe **Arithmetic**. Elle doit être une sous-classe de la classe **AbstractSeries**. Pour cette implémentation, le premier appel à la méthode **next** retournera la valeur 1.0, le second appel retournera la valeur 3.0, le troisième appel retournera la valeur 6.0, le quatrième appel retournera la valeur 10.0, et ainsi de suite. De façon générale, le $i$ème appel de la méthode retournera $S_{i-1} + i$, pour $i \in 1, 2, 3 \ldots$ et S_{i-1} est la valeur de l'appel précédent de la méthode **next**.

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

(**Question 4: suite**)

D. Vous devez implémenter la classe **Geometric**. Cette classe est une sous-classe de **AbstractSeries**. Chaque appel de la méthode **next** produit la prochaine somme partielle de cette série selon l'équation ci-bas. Le premier appel produira la valeur 1.0, le second appel produira la valeur 1.5, le troisième appel produira la valeur 1.75, etc. Vous pouvez utiliser la méthode **Math.pow(a, b)**, qui retourne a^b, pour vos calculs.

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

(**Question 4: suite**)

Les 5 premières sommes partielles de la série arithmétique sont

$S_1 = 1$

$S_2 = 1 + 2 = 3$

$S_3 = 1 + 2 + 3 = 6$

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

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

Les 5 premières sommes partielles de la série géométrique sont

$S_1 = 1$

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

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

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

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

Un appel à la méthode **next** produira la prochaine valeur de la somme partielle de cette série, c'est-à-dire la prochaine valeur S_n.

Question 5 : (25 points)

Pour cette question, vous devez compléter l'implémentation de la classe **Tuple**, l'interface **Predicate**, et la classe **IsNegative**. Voici un programme test afin de démontrer l'usage de ces classes. Le dernier énoncé affichera la valeur 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. Complétez l'implémentation de la classe **Tuple** de la page suivante. En informatique, un n-tuple est une collection linéaire de n éléments. Le type des éléments est un argument de type de la déclaration de la classe, **Tuple** est donc un type paramétré.

- L'implémentation utilise un tableau de taille fixe afin de sauvegarder les éléments de cette collection ;
- La classe possède un constructeur dont voici la signature, **Tuple(int n)**, où **n** est la taille maximale (physique) du tuple ;
- Cette classe possède une méthode **boolean add(E elem)**, où **E** est le type des éléments de cette collection. Si le tableau n'est pas rempli à capacité, la méthode ajoute l'élément à la suite des éléments de la collection et retourne **true**. Sinon, si le tuple était rempli à capacité au moment de l'appel alors la méthode retourne **false** ;
- Implémentez la méthode **int count(Predicate<E> p)**. Cette méthode retourne le nombre d'éléments de ce tuple pour lesquels le prédicat **p** retourne la valeur **true**.

(**Question 5: suite**)

```
public class Tuple               {

    // Variable(s) d'instance




    public Tuple( int n ) {









    }

    public boolean add( E elem ) {









    }

    public int count( Predicate<E> p ) {









    }

}
```

(**Question 5: suite**)

B. Complétez l'implémentation de l'interface **Predicate**. Cette interface est un type paramétré. L'interface déclare une méthode nommée **eval** ayant un paramètre dont le type est l'argument de type de l'interface. Le type de la valeur de retour de la méthode **eval** est **boolean**.

```
public                Predicate     {









}
```

C. Complétez l'implantation de la classe **IsNegative**. Cette classe réalise l'interface **Predicate**. La méthode **eval** retourne **true** si la valeur du paramètre est négative, et **false** sinon.

```
public            IsNegative                                        {









}
```

(page blanche)

(page blanche)