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 (CSI 1501)
# Examen Intra

Professeur: Marcel Turcotte

Février 2005, durée: 2 heures

## Identification

Nom, prénom: _____

Numéro d'étudiant: _____ Signature: _____

## Consignes

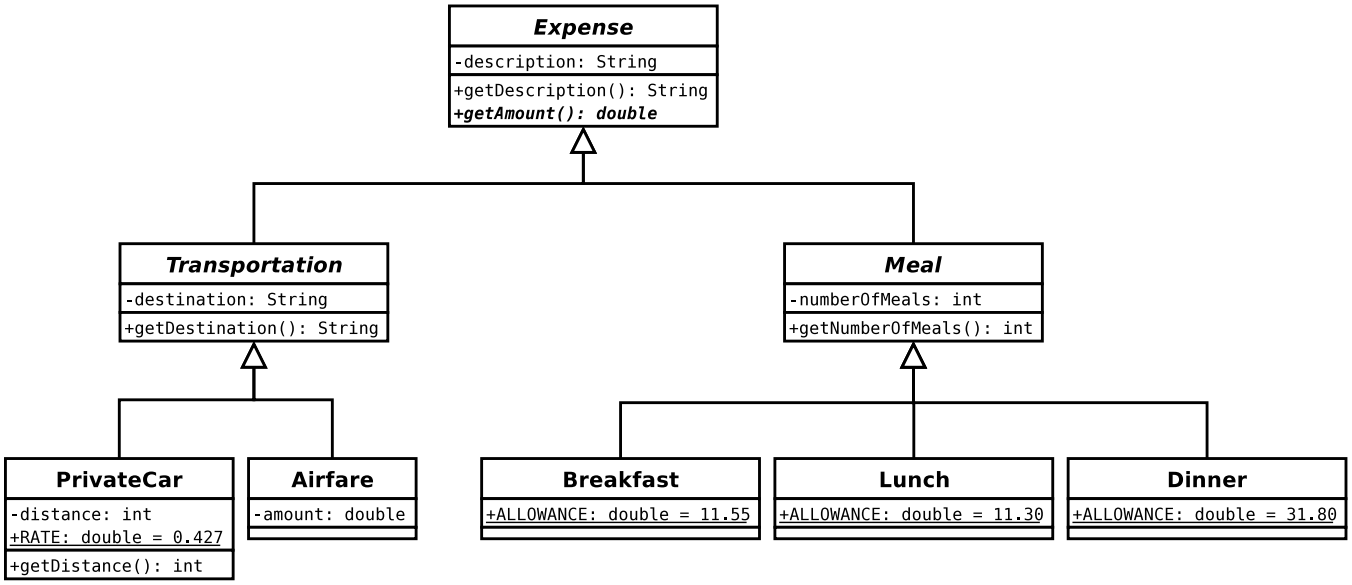
1. Livres fermés;
2. Sans calculatrice ou toute autre forme d'aide;
3. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle;
4. Écrivez lisiblement, votre note en dépend;
5. Commentez vos réponses;
6. Ne retirez pas l'agrafe.

## Barème

| Question | Maximum | Résultat |
|---:|---:|---|
| 1 | 35 | |
| 2 | 20 | |
| 3 | 15 | |
| 4 | 20 | |
| 5 | 10 | |
| **Total** | **100** | |

# Question 1: Héritage (35 points)

La compagnie "Java Financial Solutions" travaille au développement d'un nouveau logiciel pour la gestion de dépenses professionnelles. Votre rôle au sein de l'équipe de développement consiste à créer la hiérarchie de classes représant les dépenses (*Expenses*).




Spécifications:

- Toutes les dépenses ont une description (de type String);
- Toutes les dépenses liées au transport ont une destination (de type String);
- Un déplacement en voiture a une distance (de type int);
- Un déplacement par avion a un montant fixe (de type double) spécifié lors de la création d'un objet;
- Tous les repas ont un attribut représentant le nombre de repas.
- Toutes les dépenses ont une méthode qui calcule le montant total de cette dépense:
  - Le montant alloué pour un déplacement en voiture (*PrivateCar*) est le produit de la distance parcourue et d'un taux fixe (par kilomètre);
  - Le montant alloué pour un déplacement par avion (*AirFare*) est un montant fixe (spécifié au moment de la création de l'objet);
  - Le montant alloué pour les repas (*Meals*) est le produit du nombre de repas et d'un taux fixe par repas. Ce taux varie selon le type de repas (il y a trois types: Breakfast, Lunch et Dinner);

# (Question 1: suite)

Vous devez créer une implémentation en Java des classes suivantes. Assurez-vous d'y inclure les constructeurs nécessaires, les méthodes d'accès ainsi que les méthodes pour le calcul du montant total représenté par cette dépense.

**A. Expense**

**B. Transportation**

### C. PrivateCar

### D. Airfare

**E. Meal**

**F. Breakfast**

# (Question 1: suite)

**G.** Complétez l'implémentation partielle de la classe **ExpenseTracker** ci-bas. Cette classe sert à sauvegarder des dépenses. i) Ajoutez le type du tableau des éléments. ii) Complétez le constructeur. iii) Complétez l'implémentation de la méthode **double getTotal()**. La méthode **double getTotal()** retourne la somme des dépenses qui s'y trouvent.

```
public class ExpenseTracker {

    private __________[] expenses;
    private int size; // le nombre d'éléments

    public ExpenseTracker( int capacity ) {
        ___________________________________;
        size = 0;
    }
    // une méthode permettant l'ajout de nouveaux éléments
    public boolean add( Expense e ) { ... }

    public double getTotal() {









    }
}
```

Voici un programme test afin d'illustrer le type d'applications de ces classes.

```
public class Run {
    public static void main( String[] args ) {
        ExpenseTracker epro = new ExpenseTracker( 10 );
        epro.add( new PrivateCar( "ACFAS 2004", "Montréal (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() );
    }
}
```

Voici un exemple de sortie.

```
total = 508.21000000000004
```

# Question 2: Utilisation d'une pile (20 points)

Pour le devoir 2, nous avons introduit une représentation pour les interactions entre les nucléotides d'une molécule d'ARN. Ici, vous devez créer une méthode afin de compter le nombre de faux appariements (*mismatches*). Un faux appariement est une paire autre que les paires Watson-Crick (A:U,C:G,G:C or U:A) et les paires Wobble (G:U or U:G).

Voici à nouveau la description de cette représentation (chaîne d'interactions). Elle est constituée de parenthèses ouvrantes, fermantes et du point. Chaque élément de la chaîne d'interactions est associé à un élément de la chaîne de nucléotides; plus précisément ce symbole est associé au nucléotide se trouvant à la même position dans la chaîne de nucleotides. La chaîne d'interactions spécifie donc les relations entre nucléotides. Le symbole "(" représente le premier élément d'une paire. La parenthèse associée, ")", représente l'autre élément de la paire. Le point indique que le nucléotide correspondant ne participe à aucune paire. Voici un exemple, la première ligne ne fait pas partie de la représentation, elle facilite seulement l'interprétation de l'exemple.

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

Pour cet exemple, il y a deux paires canoniques, 0:8 (A:U) et 2:6 (C:G), ainsi qu'un faux appariement, 1:7 (G:A). Le nombre total de faux appariements est donc 1.

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

Pour ce deuxième exemple, il y a trois faux appariements, 4:68 (A:G), 28:42 (U:U), 53:61 (G:G). Le nombre total de faux appariements est 3.

- Vous devez écrire la méthode **countMismatches( String nucs, String pairs )** pour la classe **Rna** ci-bas;
- L'algorithme doit utiliser une pile ainsi qu'une stratégie semblable à celle du devoir 2;
- Assumez l'existence d'une classe nommée **LinkedStack** réalisant l'interface **Stack**.

# (Question 2: suite)

```
public class Rna {

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

  } // fin de countMismatches
} // fin de Rna
```

# Question 3: LinkedStack (15 points)

À l'intérieur de la classe **LinkedStack** ci-bas, implémentez la méthode **Object[] toArray()**. **LinkedStack** réalise l'interface **Stack** et utilise une structure chaînée afin de sauvegarder un nombre illimité d'éléments.

La méthode **Object[] toArray()** retourne un tableau *i)* dont la taille correspond au nombre d'éléments dans la pile et *ii)* contenant les mêmes éléments. L'élément du dessous doit être sauvegardé en position 0, le second élément, en comptant à partir du fond de la pile, sera sauvegardé en position 1, etc. De sorte que l'élément du dessus sera en position **size()-1**. La pile doit demeurer inchangée suite à un appel à la méthode toArray.

Voici un programme test afin de vous aidez à mieux comprendre le travail demandé pour cette question.

```
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 ] );
        }
    }
}
```

Son exécution produira la sortie suivante.

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

- **LinkedStack** possède une méthode **int size()** retournant le nombre d'éléments se trouvant présentement dans la pile;
- **LinkedStack** réalise toutes les méthodes de l'interface **Stack**, cependant, vous ne pouvez pas les utiliser, vous devez "traverser" la pile et copier les éléments un à un;
- La méthode **Object[] toArray()** ne doit pas changer l'état de la pile (i.e. suite à un appel à la méthode toArray, la pile doit contenir les mêmes éléments, dans le même ordre, qu'avant l'appel);
- Vous pouvez utiliser la méthode **int size()** pour votre implémentation de la méthode **Object[] toArray()**.

# (Question 3: suite)

```
public class LinkedStack implements Stack {

    private static class Elem { // Noeuds de la structure chaînée
        private Object value;
        private Elem next;
        private Elem( Object value, Elem next ) {
            this.value = value;
            this.next = next;
        }
    }

    private Elem topElem; // Variables d'instance
    private int size;

    public int size() { return size; }

    // Les autre méthodes seraient ici.

    public Object[] toArray() {  // Complétez l'implémentation









    } // fin de toArray
} // fin de LinkedStack
```

# Question 4: Structure de données générique (20 points)

Définissons une structure de données indexable, c'est-à-dire que l'on peut accéder par position. Cette structure de données, nommée **Sequence**, peut sauvegarder des éléments de type **Object**. Tel un tableau, un index (de type int) est utilisé afin d'accéder à un élément de la séquence. À la différence des tableaux, le plus petit index n'est pas nécessairement 0. Les bornes des indexes sont données au moment de la création d'une séquence. Par exemple, "**Sequence ss = new Sequence( 100, 200 )**" va créer une nouvelle Sequence afin de sauvegarder 101 objets et l'index du premier élément sera 100 alors que celui du dernier sera 200. Aussi contrairement au tableaux traditionnels, une séquence peut changer sa taille dynamiquement. Vous trouverez une implémentation partielle de cette classe sur la page qui suit.

**A.** (8 points) Effectuez tous les changements nécessaires à la classe Sequence afin que des exceptions soient lancées lorsque les paramètres ne sont pas valides.

- Lors de la création d'une nouvelle séquence, on doit lancer une exception de type IllegalArgumentException si la borne inférieure est plus grande que la borne supérieure;
- Une exception de type IndexOutOfBoundsException sera lancée pour tout accès à une position inexistante, donc à l'extérieur de l'intervalle des positions valides;
- La valeur null est considérée comme une valeur illégale. Vous devez lancer une exception de type IllegalArgumentException si l'utilisateur ajoute la valeur null à la séquence;
- Lorsqu'on accroît la borne supérieure le nouvel index doit être plus grand que l'index supérieur courant, sinon lancez une exception de type IllegalArgumentException;
- IllegalArgumentException et IndexOutOfBoundsException sont des sous-classes de RuntimeException.

**B.** (12 points) Vous devez implémenter la méthode **increaseHigh( int newIndex )** qui permet d'accroître la capacité de la Sequence. Avant un appel à cette méthode, la capacité de la séquence est de **high** - **low** + 1 éléments. Après un appel à la méthode increaseHigh, cette séquence peut contenir **newIndex** - **low** + 1 éléments.

Effectuez tous vos changements directement dans la classe Sequence ci-bas. Utilisez des étiquettes ou des flèches, si nécessaire, afin d'indiquer clairement l'endroit dans le code où les changements doivent être effectués.

```
public class Sequence {

    // variables d'instance

    private Object[] elems; // utilisé pour stocker les élément de cette Sequence

    private int low;        // borne inférieure
    private int high;       // borne supérieure

    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 newHigh ) {

        // complétez l'implémentation









    } // fin de increaseHigh
} // fin de Sequence
```

# Question 5: Réponses brèves (10 points)

**A.** (4 points) Pour cette question nous avons ajouté les méthodes **findAndReplace** et **main** à la classe Sequence de la question 4. Cette nouvelle implémentation se trouve à la page suivante. Parmi les alternatives suivantes, laquelle correspond à l'exécution de la méthode principale (main).

(a) Lance 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) Aucune de ces réponses.

**B.** (6 points) Utilisez l'algorithme à base de pile vu en classe afin d'évaluer l'expression postfixe suivante et montrez le contenu de la pile des opérandes avant et après le traitement de chaque opérateur (il y a donc 8 piles).

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

# (Question 5: suite)

```
public class Sequence {

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

    // Les autres méthodes de la classe Sequence seraient insérées ici.

    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 ));

    }
}
```

# (page blanche)