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 2007, durée: 2 heures

Identification

Nom, prénom : _____

Numéro d'étudiant : _____ Signature : _____

Consignes

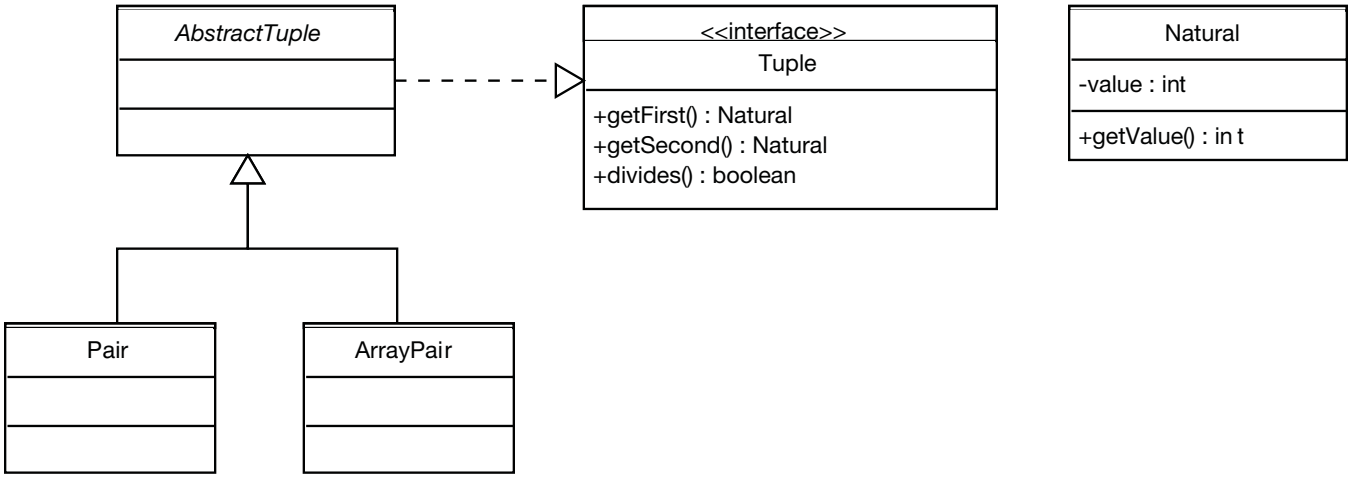
1. Livres fermés ;
2. Sans calculatrice ou toute autre forme d'aide ;
3. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle ;
4. Écrivez lisiblement, votre note en dépend ;
5. Commentez vos réponses ;
6. Ne retirez pas l'agrafe.

Barème

Question	Maximum	Résultat
1	36	
2	15	
3	15	
4	12	
5	22	
Total	**100**	

Question 1 : (36 points)

Un tuple permet de sauvegarder deux nombres naturels (des objets de la classe **Natural**). Tous les tuples ont une méthode **getFirst** ainsi qu'une méthode **getSecond** retournant une référence vers le premier et le second nombre naturel du tuple, respectivement. Un tuple possède une méthode **divides** qui retourne la valeur **true** si le premier élément est un diviseur du second élément, et **false** sinon.



Pour cette question, il y a une interface nommée **Tuple**, une classe abstraite nommée **AbstractTuple**, ainsi que deux implémentations concrètes, nommées **Pair** et **ArrayPair**. Vous trouverez leur description complète dans les pages qui suivent. L'implémentation de la classe **Natural** se retrouve à la page 18. L'exécution des énoncés ci-dessous produira la sortie suivante : `7 is a factor of 42`.

```
Natural n1, n2, n3;

n1 = new Natural( 7 );
n2 = new Natural( 17 );
n3 = new Natural( 42 );

Tuple t1, t2;

t1 = new Pair( n1, n2 );
t2 = new ArrayPair( n1, n3 );

if ( t1.divides() ) {
    System.out.println( t1.getFirst() + " is a factor of " + t1.getSecond() );
}

if ( t2.divides() ) {
    System.out.println( t2.getFirst() + " is a factor of " + t2.getSecond() );
}
```

Assurez-vous d'implémenter tous les constructeurs et toutes les méthodes d'accès nécessaires à l'exécution des énoncés ci-dessus.

A. Donner l'implémentation de l'interface **Tuple**. Cette interface déclare 3 méthodes. Il y a deux méthodes d'accès, nommées **getFirst** et **getSecond**, qui toutes deux retournent une référence de type **Natural**. Finalement, l'interface déclare aussi une méthode nommée **divides** qui retourne un booléen. (8 points)

B. Concevoir la classe abstraite **AbstractTuple**. Elle réalise l'interface **Tuple**. Cette classe fournit une implémentation concrète de la méthode **divides**, qui retourne la valeur **true** si le premier nombre du tuple est un diviseur du second, et **false** sinon. Un diviseur de n est une nombre qui divise n sans reste. (8 points)

C. Créer une implémentation concrète de la classe **AbstractTuple** que vous nommerez **Pair**. Cette implémentation possède deux variables d'instance. Ces variables sont des références vers le premier et le second nombre du tuple. Ajoutez tous les constructeurs et toutes les méthodes d'accès que vous jugez nécessaires. (10 points)

D. Vous devez concevoir une implémentation concrète de la classe **AbstractTuple** que vous nommerez **ArrayPair**. L'implémentation utilise un tableau de taille 2 afin de sauvegarder les références vers le premier et le second nombre du tuple. Ajoutez tous les constructeurs et toutes les méthodes d'accès que vous jugez nécessaires. (10 points)

Question 2 : (15 points)

Complétez l'implémentation de la méthode statique **boolean isPalindrome(CharReader reader)**. Définissons un **palindrome** comme étant une phrase qui peut être lue indifféremment de gauche à droite ou de droite à gauche lorsqu'on ignore les ponctuations, accents et espaces. Voici des exemples de palindromes :

- i prefer pi
- never odd or even
- was it a cat i saw

Pour cette question, les données sont obtenues à l'aide d'un objet **CharReader**. Ainsi, vous ne connaissez pas la taille de l'entrée et il est impossible de relire les données qui ont été lues. Vous devez vous conformer aux consignes qui suivent.

- **boolean isPalindrome(CharReader reader)**. Implémentez un algorithme à base de pile qui retourne **true** si toute la phrase spécifiée par l'objet **reader** est un palindrome selon la définition ci-haut, et **false** sinon ;
- Le paramètre de la méthode est de type **CharReader**. Un objet de la classe **CharReader** possède **exactement** deux méthodes d'instance (et aucune variable publique). Une fois les données lues, il est impossible de les lire à nouveau :
  - **boolean hasMoreChars()** ; retourne **true** s'il y a des caractères supplémentaires à retourner, c'est-à-dire si le prochain appel à **nextChar()** pourra être complété avec succès, et **false** sinon ;
  - **char nextChar()** ; retourne le prochain caractère de l'entrée.
- Vous ne devez pas utiliser de tableaux ou de chaînes de caractères afin de sauvegarder les données qui sont lues, vous devez utiliser des piles ;
- Supposez l'existence d'une classe **StackImpl** qui réalise l'interface **Stack**. Pour cette question, les éléments sauvegardés sont des caractères.

```
public interface Stack {
    public abstract boolean isEmpty();
    public abstract char peek();
    public abstract char pop();
    public abstract void push( char element );
}
```

- **StackImpl** permet de sauvegarder un nombre illimité de caractères ;
- **Character.isLetter(char ch)** retourne **true** si le caractère **ch** représente une lettre, et **false** sinon.

```
public static boolean isPalindrome( CharReader reader ) {

}
```

Question 3 : (15 points)

La classe **DynamicArrayStack** ci-dessous utilise la technique vue en classe, ainsi qu'au devoir 3, afin d'accroitre ou de diminuer la taille physique de la pile selon les besoins de l'application.

- **DynamicArrayStack** utilise un tableau afin de sauvegarder les éléments de cette pile ;
- L'interface **Stack** et sont implémentation, **DynamicArrayStack**, ont un paramètre formel de type (interface et classe paramétrée à l'aide du concept *generics* introduit par la version 1.5 de Java) ;
- La capcité initiale du tableau est spécifiée à l'aide du premier paramètre du constructeur ;
- La taille physique du tableau s'accroit par un incrément fixé (**increment**) lorsque la méthode **void push(E elem)** détecte que le tableau est plein ;
- La taille physique du tableau décroit d'un nombre fixé de cellules (**increment**) lors d'un appel à la méthode **E pop()** si le nombre de cellules libres est **increment** ou plus ;
- La valeur de l'incrément (**increment**) est spécifiée à l'aide du second paramètre du constructeur ;
- La variable d'instance **top** désigne la position de l'élément du dessus, ou -1 si la pile est vide.

A. Corrigez au moins 5 erreurs de compilation ou d'exécution de l'implémentation partielle ci-dessous. (5 points)

B. Complétez l'implémentation partielle de la classe **DynamicArrayStack** à l'aide des informations ci-dessus. (10 points)

```
public class DynamicArrayStack<E> implements Stack<E> {

    // Variables d'instance
    private static E[] elems;    // Sauvegarde les éléments de cette pile
    private static int top = -1; // Désigne l'élément du dessus
    private final int capacity;  // Mémorise la capacité initiale
    private final int increment; // Utilisé afin de changer la taille physique

    public DynamicArrayStack( int capacity, int increment ) {
        E[] elems = (E[]) new Object[ capacity ];
        this.capacity = capacity;
        this.increment = increment;
    }

    // Retourne true si cette pile est vide
    public boolean isEmpty() {
        return top == 0;
    }

    // Se poursuit à la page qui suit ...
```

DynamicArrayStack (suite)

```
public void push( E element ) {

    if ( ___________________________ ) {
        increaseSize();
    }

    elems[ top ] = element;
    top++;
}

private void increaseSize() {

    E[] newElems;
    int newSize;

    newSize= elems.length + increment;

    newElems = new E[ newSize ];

    for ( int i=0; i<elems.length; i++ ) {
        newElems[ i ] = elems[ i ];
    }

    ______________________________;
}

// Se poursuit à la page qui suit ...
```

DynamicArrayStack (suite)

```
    public E peek() {

        return ______________________________;
    }

    // Complétez l'implémentation de pop()

    public E pop() {


        E saved;
        saved = elems[ top ];


        elems[ top ] = ________________;


        top--;


        return saved;
    }


    private void decreaseSize() {

        E[] newElems;
        int newSize;

        newSize = elems.length - increment;

        if ( newSize < capacity ) {
            newSize = capacity;
        }

        ________________________________________;

        for ( int i=0; i<=top; i++ ) {
            newElems[ i ] = elems[ i ];
        }

        elems = newElems;
    }

} // Fin de DynamicArrayStack
```

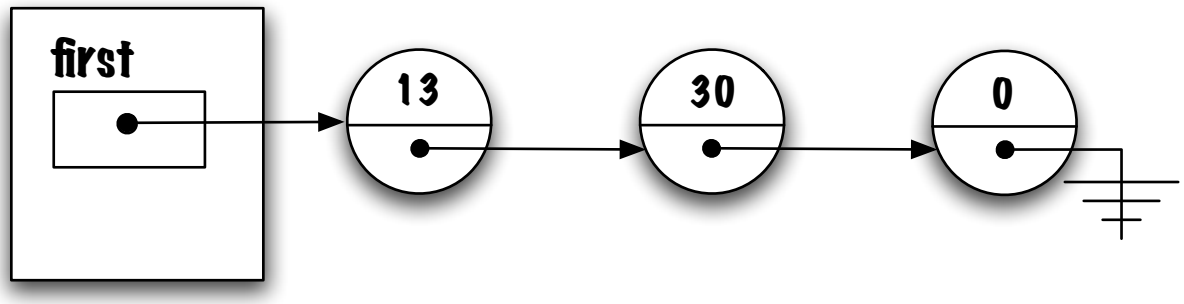
DynamicArrayStack (suite)

Complétez la méthode principale (**main**) de la classe **Test** ci-dessous. Cette méthode déclare une référence vers une pile d'objets de la classe **Integer**, crée une pile qui servira à sauvegarder des objets de la classe **Integer**, ajoute 20 éléments sur la pile, retire et imprime ces éléments.

```
public class Test {

    public static void main( String[] args ) {

        // Déclarer une référence vers une pile d'objets Integer

        _____________ s;

        // Créer une instance de DynamicStack pour sauvegarder des objets Integer

        s = ______________________________________;

        for ( int i=0; i<20; i++ ) {
            s.push( new Integer( i ) );
        }

        while ( ! s.isEmpty() ) {
            // Déclarer un Integer

            _______________ elem;


            // Retirer un élément de la pile


            elem = _____________;

            System.out.println( elem );
        }
    }

} // Fin de Test
```

Question 4 : (12 points)

Complétez l'implémentation de la classe **Time** ci-dessous. Des éléments chaînés les uns aux autres sont utilisés afin de sauvegarder les heures, minutes et secondes de cette valeur de temps. En particulier, la classe **Time** n'a qu'une seule variable d'instance, nommée **first**. Il s'agit d'une référence vers un objet **Elem** servant à sauvegarder les heures de cette valeur de temps. La variable **next** de cet objet désigne un objet **Elem** servant à sauvegarder le nombre de minutes de cette valeur de temps. Sa variable **next** désigne un objet de la classe **Elem** servant à sauvegarder les secondes de ce temps. Par exemple, la création de cet objet : **new Time(13, 30, 0)**, aura le diagramme de mémoire suivant.



Ici, les valeurs sauvegardées dans un objet de la classe **Elem** sont du type **int** (un type primitif).

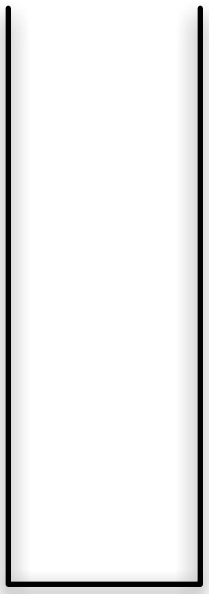
```
public class Time {
    private static class Elem {
        private int value;
        private Elem next;
        private Elem( int value, Elem next ) {
            this.value = value;
            this.next = next;
        }
    }
    private Elem first;

    public Time( int hours, int minutes, int seconds ) {









    }
```

```
    public int getHours() {

    }

    public int getMinutes() {

    }

    public int getSeconds() {

    }
} // Fin de Time
```

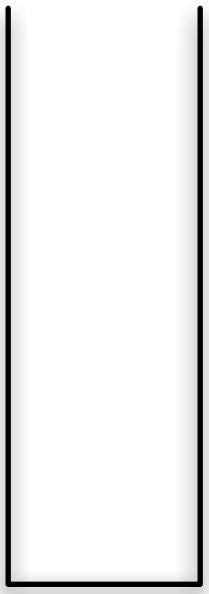
Question 5 : (22 points)

A. Utilisez l'algorithme à base de pile vu en classe afin d'évaluer l'expression postfixée (RPN) suivante. Donnez le contenu de la pile immédiatement avant et après l'évaluation de chaque opérateur. La notation $-_x$ désigne la première, deuxième et troisième soustraction de l'expression. (8 points)

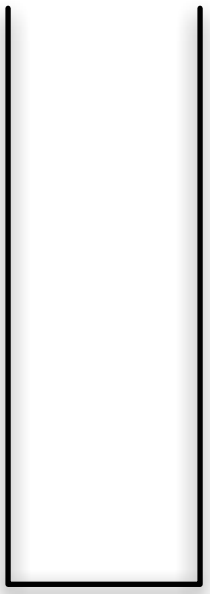
$$8 \quad 2 \quad -_a \quad 10 \quad 8 \quad 1 \quad -_b \quad -_c \quad /$$



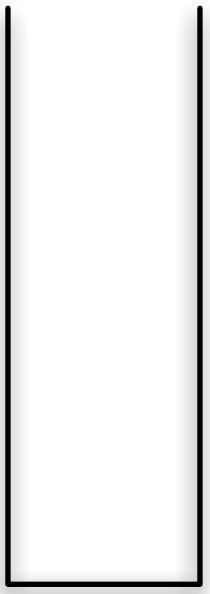
Avant $-_a$



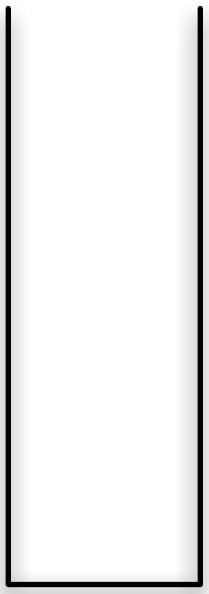
Après $-_a$



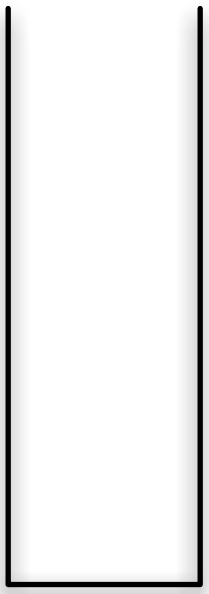
Avant $-_b$



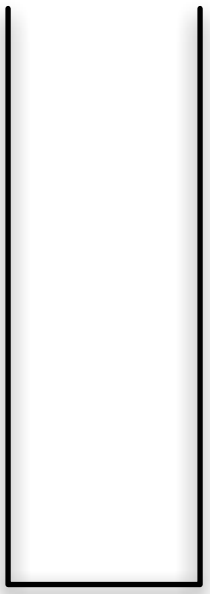
Après $-_b$



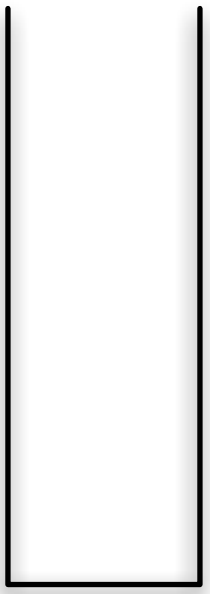
Avant $-_c$



Après $-_c$



Avant /



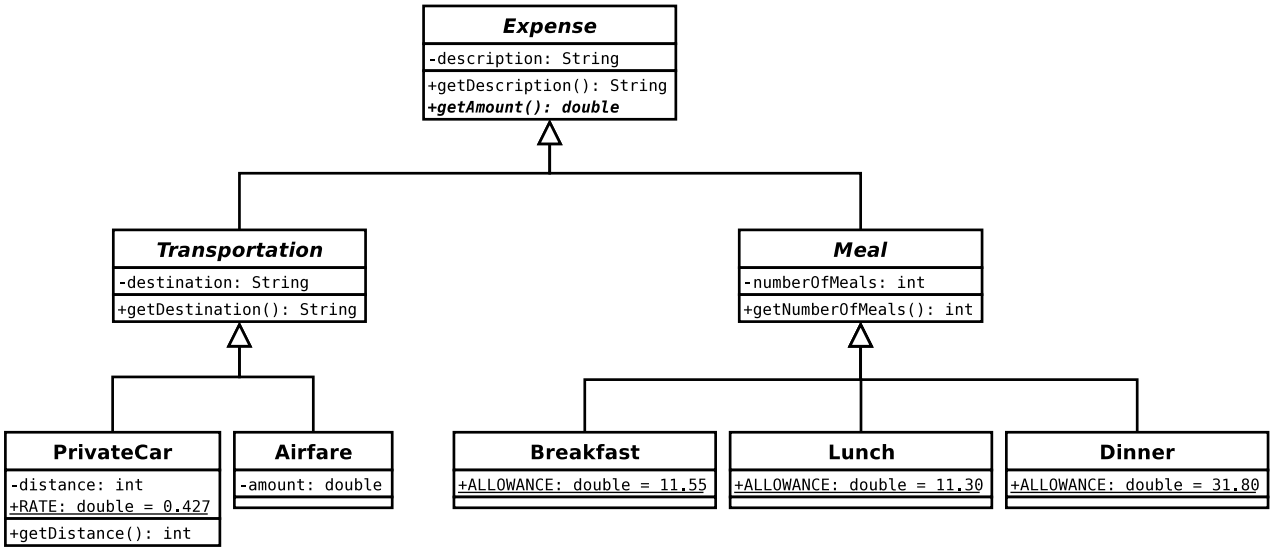
Après /

B. Représentez l'expression en notation infixée suivante à l'aide de la notation postfixée (notation polonaise inverse) (4 points)

$$15 - 3 * 5 + 8/2$$

Solution :

C. La hiérarchie de classes suivante est tirée d'un système informatique servant au suivi de dépenses professionnelles.



Specification :

- Toutes les dépenses ont une description (une chaîne de caractères) ;
- **Transportation** et **Meal** sont des sous-classes de la classe **Expense** ;
- **Expense**, **Transportation** et **Meal** sont des classes abstraites ;
- **PrivateCar** et **Airfare** sont des sous-classes de la classe **Transportation** ; **Breakfast**, **Lunch** et **Dinner** sont des sous-classes de la classe **Meal** ;
- Toutes dépenses liées au transport ont une destination (une chaîne de caractères) ;
- La classe **Transportation** réalise l'interface **Taxable** (non-illustrée sur le diagramme ci-haut) ;
- L'interface **Taxable** déclare la méthode **double getTax()** ;
- 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 servant au calcul du 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) ;

Quels énoncés sont valides ? (10 points)

(a) `Transportation t = new Transportation( "ISMB 2007", "San Fransisco" ) ;`

(b) `Expense e = new Transportation( "ISMB 2007", "San Fransisco" ) ;`

(c) `Taxable t = new Transportation( "ISMB 2007", "San Fransisco" ) ;`

(d)
```
Expense e = new Airfare( "ISMB 2007", "San Fransisco", 789.0 ) ;
double a = e.getAmount()
```

(e) `Taxable t = new Airfare( "ISMB 2007", "San Fransisco", 789.0 ) ;`

(f)
```
Taxable t = new Airfare( "ISMB 2007", "San Fransisco", 789.0 ) ;
double x = t.getTax() ;
```

(g)
```
Expense e = new Airfare( "ISMB 2007", "San Fransisco", 789.0 ) ;
double x = e.getTax() ;
```

(h) **Taxable t = new Taxable() ; double x = t.getTax() ;**

(i)
```
Expense e = new Airfare( "ISMB 2007", "San Fransisco", 789.0 ) ;
String s = e.getDestination() ;
```

(j)
```
Expense t = new Airfare( "ISMB 2007", "San Fransisco", 789.0 ) ;
double x = t.getAmount() ;
```

A Natural

```
/* La classe Natural encapsule une valeur du type primitif int dans
 * un objet.  De plus, la valeur est plus grande ou égale à zéro.
 */

public class Natural {

    private int value; // variable d'instance

    /* Constructeur de la classe Natural.  Initialise la valeur de
     * cet objet.  Lance l'exception IllegalArgumentException si
     * la valeur du paramètre est négative.
     */

    public Natural( int value ) {
        if ( value < 0 ) {
            throw new IllegalArgumentException( "less than zero" );
        }
        this.value = value;
    }

    /* Retourne la valeur de cet objet.
     */

    public int getValue() {
        return value;
    }

    /* Retourne une chaîne représentant cet objet.
     */

    public String toString() {
        return Integer.toString( value );
    }
}
```