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 2008, durée: 2 heures

Identification

Nom, prénom : _____

Numéro d'étudiant : _____ Signature : _____

Consignes

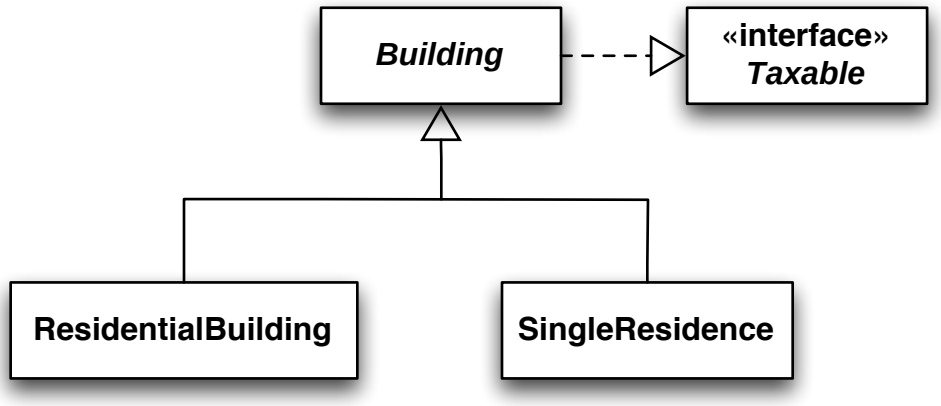
1. Livres fermés ;
2. Sans calculatrice ou toute autre forme d'aide ;
3. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle ;
4. Écrivez lisiblement, votre note en dépend ;
5. Commentez vos réponses ;
6. Ne retirez pas l'agrafe.

Barème

Question	Maximum	Résultat
1	35	
2	10	
3	10	
4	10	
5	10	
Total	**75**	

Question 1 : (35 points)

Pour cette question, il y a une interface, nommée **Taxable**, ainsi que trois classes, **Building** (immeuble), **ResidentialBuilding** (immeuble d'habitation) et **SingleResidence** (maison unifamiliale).



Sachant que :

1. tous les immeubles ont une adresse (une chaîne de caractères) ;
2. tous les immeubles ont une valeur d'évaluation (double) ;
3. les immeubles d'habitation ont un nombre d'unités locatives (un entier) ;
4. les résidences unifamiliales ont une piscine ou pas (un booléen) ;
5. tous les immeubles sont taxable ; il y a une interface nommée **Taxable** et elle déclare une méthode nommée **getTaxAmount** ;
6. le montant de la taxe pour un immeuble d'habitation est la somme de deux valeurs. Premièrement, il y a une taxe générale, c'est le produit d'un taux fixe (0.75%) par le montant de l'évaluation. Deuxièmement, il y a une taxe pour le ramassage des ordures, c'est le produit d'un montant fixe (149.77 $) par le nombre d'unités locatives ;
7. le montant de la taxe pour les maisons unifamiliales est la somme des valeurs suivantes : premièrement, la taxe générale, qui est le produit d'un taux fixe (0.8%) et du montant de l'évaluation de la propriété. Deuxièmement, il y a la taxe pour le ramassage des ordures, c'est un montant fixe (159.77 $). Finalement, il y a une taxe de $89.01 pour la possession d'une piscine.

Implémentez, en Java, les trois classes décrites ci-dessus, ainsi que l'interface. Vous devez inclure tous les constructeurs nécessaires. Déclarez des constantes lorsque c'est nécessaire. Pour tous les attributs, créer des méthodes d'accès. Cependant, les objets doivent être immuables (*immutable*) ; un concept vu en classe au cours 10.

A. Implémentez, en Java, les classes décrites précédemment, ainsi que l'interface. (27 points)

B. Créez une méthode de classe, nommée **tallyTaxes**. Son premier paramètre est une référence vers un tableau d'objets **Taxable**. Le second paramètre représente le nombre de cellules occupées du tableau. Faites l'hypothèse que i) la valeur de ce paramètre sera toujours inférieure ou égale au nombre de cellules du tableau, ii) les éléments seront toujours sauvegardés dans des cellules contiguës, à la gauche du tableau. La méthode retourne la somme des taxes de tous les objets sauvegardés dans ce tableau. (5 points)

C. Déclarez et créez un tableau, nommé **city**, afin de contenir 23,000 immeubles. (3 points)

Question 2 : Utilisation d'une pile (10 points)

La classe **Calculator** (page suivante) implémente un interprète pour des expressions en notation postfixe semblable à celui étudié en classe. Donnez le contenu de la sortie suite à l'exécution des deux énoncés suivants :

```
Calculator c = new Calculator();
c.execute( "6 4 5 print * print 2  3 print ^ print ~ print + print" );
```

Notes :

- Pour cette question, la classe **LinkedStack** réalise l'interface **Stack** ;
- Vous trouverez l'implémentation des classes **Token** et **Reader** aux sections B et C.

Utilisation d'une pile (suite)

```
public class Calculator {
    private Stack<Token> operands = new LinkedStack<Token>();
    public void execute( String program ) {
        Reader r = new Reader( program );
        while ( r.hasMoreTokens() ) {
            Token t = r.nextToken();
            if ( ! t.isSymbol() ) {
                operands.push( t );
            } else if ( t.sValue().equals( "+" ) ) {
                Token b = operands.pop();
                Token a = operands.pop();
                Token res = new Token( a.iValue() + b.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "-" ) ) {
                Token b = operands.pop();
                Token a = operands.pop();
                Token res = new Token( a.iValue() - b.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "*" ) ) {
                Token b = operands.pop();
                Token a = operands.pop();
                Token res = new Token( a.iValue() * b.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "~" ) ) {
                Token o = operands.pop();
                Token res = new Token( - o.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "^" ) ) {
                Token b = operands.pop();
                Token a = operands.pop();
                Token res = new Token( (int) Math.pow( a.iValue(), b.iValue() ) );
                operands.push( res );
            } else if ( t.sValue().equals( "print" ) ) {
                System.out.println( "-top-" );
                Stack<Token> tmpStack = new LinkedStack<Token>();
                while ( ! operands.isEmpty() ) {
                    t = operands.pop();
                    System.out.println( t );
                    tmpStack.push( t );
                }
                while (! tmpStack.isEmpty()) {
                    operands.push( tmpStack.pop() );
                }
                System.out.println( "-bottom-" ); System.out.println();
            }
        }
    }
}
```

Question 3 : (10 points)

La classe **DynamicArrayStack** ci-dessous utilise la technique vue en classe afin d'accroître 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 son implémentation, **DynamicArrayStack**, ont un paramètre formel de type (types génériques introduit à partir de la version 1.5 de Java) ;
- La capacité initiale du tableau est spécifiée par la constante **INITIAL_CAPACITY** ;
- La taille physique du tableau s'accroît par un facteur (constante **FACTOR**) lorsque la méthode **void push(E elem)** détecte que le tableau est plein ;
- La variable d'instance **top** désigne la position de l'élément du dessus, ou -1 si la pile est vide.

A. Complétez l'implémentation de méthode **compact**. La méthode diminue la taille physique du tableau afin qu'il n'y ait aucune cellule inoccupée. Cependant, la taille physique du tableau n'est jamais moins de **INITIAL_CAPACITY**.

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

    // Constantes

    public static final int INITIAL_CAPACITY = 10;
    public static final double FACTOR = 1.5;

    // Variables d'instance

    private E[] elems;           // Pour la sauvegarde des éléments
    private int top = -1;        // Désigne l'élément du dessus

    public DynamicArrayStack() {
        elems = (E[]) new Object[ INITIAL_CAPACITY ];
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public E peek() {
        return elems[ top ];
    }

    // Se poursuit...
```

```
    public void push( E element ) {
        if ( ( top + 1 ) == elems.length ) {
            increaseSize();
        }
        top++;
        elems[ top ] = element;
    }

    private void increaseSize() {
        // L'implémentation de la méthode increaseSize serait ici. Elle a
        // cependant été retirée pour éviter de communiquer trop d'informations.
    }

    public E pop() {
        E saved;
        saved = elems[ top ];
        elems[ top ] = null;
        top--;
        return saved;
    }

    public void compact() { // Complétez cette méthode

    } // Fin de compact
} // Fin de DynamicArrayStack
```

Question 4 : (10 points)

Complétez l'implémentation des méthodes d'instance **size()** et **swap()** pour la classe **LinkedStack** ci-bas.

A. La méthode **size()** retourne le nombre d'éléments se trouvant dans la pile ;

B. La méthode **swap** inverse l'ordre des deux premiers éléments, c'est-à-dire que le premier élément devient le deuxième et le deuxième devient le premier ; **la méthode doit échanger l'ordre des éléments et non leurs valeurs**. Vous ne devez pas utiliser les méthodes **push** et **pop**, vous devez manipuler les liens (références) directement. La méthode **swap** retourne **false** si la pile contient moins de deux éléments.

```
public class LinkedStack<T> implements Stack<T> {

    private static class Elem<E> { // Les noeuds de la liste
        private E info;
        private Elem<E> next;
        private Elem( E info, Elem<E> next) {
            this.info = info;
            this.next = next;
        }
    }

    private Elem<T> top; // Variable d'instance, désigne l'élément du dessus

    public int size() {









    }

    // Se poursuit...
```

```
    public boolean swap() {









    } // Fin de swap

    // Les autres méthodes d'instance, telles que push et pop, serait ici,
    // vous ne devez pas les utiliser.

} // Fin de LinkedStack
```

Question 5 : (10 points)

A. Donnez l'expression postfixe (notation polonaise inverse) correspondant à l'expression infixe suivante.

$$((5 + 3) / 2) \times (4 - 1)$$

B. Pour cette question, il y a l'interface **Valuable**, la classe **Number** et la classe **Integer**, telles que définies à la page 14.

Lequel des 4 résultats suivants caractérise le mieux chacun des 8 tests se trouvant à la page qui suit.

(**1**) Produit une erreur de compilation ;

(**2**) Produit une erreur d'exécution ;

(**3**) Affiche «if» sur la sortie standard ;

(**4**) Affiche «else» sur la sortie standard ;

Pour chaque test, encerclez une seule réponse.

(a) **1 2 3 4**

(b) **1 2 3 4**

(c) **1 2 3 4**

(d) **1 2 3 4**

(e) **1 2 3 4**

(f) **1 2 3 4**

(g) **1 2 3 4**

(h) **1 2 3 4**

(a)
```
Number n;

n = new Integer( 3 );

if ( n.getType().equals( "Number" ) ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

(b)
```
Valuable v;

v = new Valuable( 3 );

if ( v.getValue() == 3 ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

(c)
```
Number n;

Integer i;

n = new Integer( 3 );

i = n;

if ( n.getValue() == 3 ) {
     System.out.println( "if" );
} else {
     System.out.println( "else" );
}
```

(d)
```
Integer i, j, k;

i = new Integer( 1 );
j = new Integer( 2 );

k = i.plus( j );

if ( k.getValue() == 3 ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

(e)
```
Object o = new String( "3" );

Integer i;

i = (Integer) o;

if ( i.getValue() == 3 ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

(f)
```
Number n;

n = new Integer( 3 );

if ( n.toString().equals( "Integer: 3" ) ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

(g)
```
Integer i, j;

i = new Integer( 3 );
j = new Integer( 3 );

if ( i == j ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

(h)
```
Integer i, j;

i = new Integer( 3 );
j = new Integer( 3 );

if ( i.equals( j ) ) {
    System.out.println( "if" );
} else {
    System.out.println( "else" );
}
```

Déclarations pour la Question 5: de la page 12.

```
public interface Valuable {
    public int getValue();
}

public abstract class Number implements Valuable {

    public String toString() {
        return "Number: " + getValue();
    }

    public String getType() {
        return "Number";
    }
}

public class Integer extends Number {

    private int value;

    public Integer( int value ) {
        this.value = value;
    }

    public int getValue () {
        return value;
    }

    public Integer plus( Integer other ) {
        int value = 0;
        return new Integer( value + other.value );
    }

    public String toString() {
        return "Integer: " + getValue();
    }

    public String getType() {
        return "Integer";
    }
}
```

A Stack

```
public interface Stack<T> {

    /**
     * Vérifie si cette pile est vide.
     *
     * @return true si cette pile est vide; et false sinon.
     */

    public abstract boolean isEmpty();

    /**
     * Retire et retourne l'élément du dessus
     *
     * @return l'élément du dessus
     */

    public abstract T pop();

    /**
     * Ajoute un élément sur la pile.
     *
     * @param l'élément à ajouter sur la pile.
     */

    public abstract void push( T element );

}
```

B Token

```
public class Token {
    private static final int INTEGER = 1;
    private static final int SYMBOL = 2;

    private int iValue;
    private String sValue;

    private int type;

    public Token( int iValue ) {
        this.iValue = iValue;
        type = INTEGER;
    }
    public Token( String sValue ) {
        this.sValue = sValue;
        type = SYMBOL;
    }
    public int iValue() {
        // pré-condition: ce jeton représente un entier
        return iValue;
    }
    public String sValue() {
        // pré-condition: ce jeton représente un symbole
        return sValue;
    }
    public boolean isInteger() {
        return type == INTEGER;
    }
    public boolean isSymbol() {
        return type == SYMBOL;
    }
    public String toString() {
        switch ( type ) {
        case INTEGER:
            return "INTEGER: " + iValue;
        case SYMBOL:
            return "SYMBOL: " + sValue;
        default:
            return "INVALID";
        }
    }
}
```

C Reader

```
import java.util.StringTokenizer;

public class Reader {

    private StringTokenizer st;

    public Reader( String s ) {
        st = new StringTokenizer( s );
    }
    public boolean hasMoreTokens() {
        return st.hasMoreTokens();
    }

    public Token nextToken() {

        String t = st.nextToken();

        try {

            return new Token( Integer.parseInt( t ) );

        } catch ( NumberFormatException e ) {

            return new Token( t );

        }
    }
}
```

(page blanche)