Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Introduction à l'informatique II (ITI1521)

EXAMEN DE MI-SESSION

Instructeurs: Guy-Vincent Jourdan et Marcel Turcotte

Mars 2018, durée: 2 heures

Identification

Nom de famille : _____ Prénom(s) : _____

Étudiant : _____ Signature : _____

Instructions

1. Examen à livres fermés.
2. L'utilisation de calculatrices, d'appareils électroniques ou tout autre dispositif de communication est interdit.
   (a) Tout appareil doit être éteint et rangé.
   (b) Toute personne qui refuse de se conformer à ces règles pourrait être accusée de fraude scolaire.
3. Répondez sur ce questionnaire.
   (a) Utilisez le verso des pages si nécessaire.
   (b) Aucune page supplémentaire n'est permise.
4. Écrivez vos commentaires et hypothèses afin d'obtenir des points partiels
5. Écrivez lisiblement, puisque votre note en dépend.
6. Ne retirez pas l'agrafe du livret d'examen.
7. Attendez l'annonce de début de l'examen.

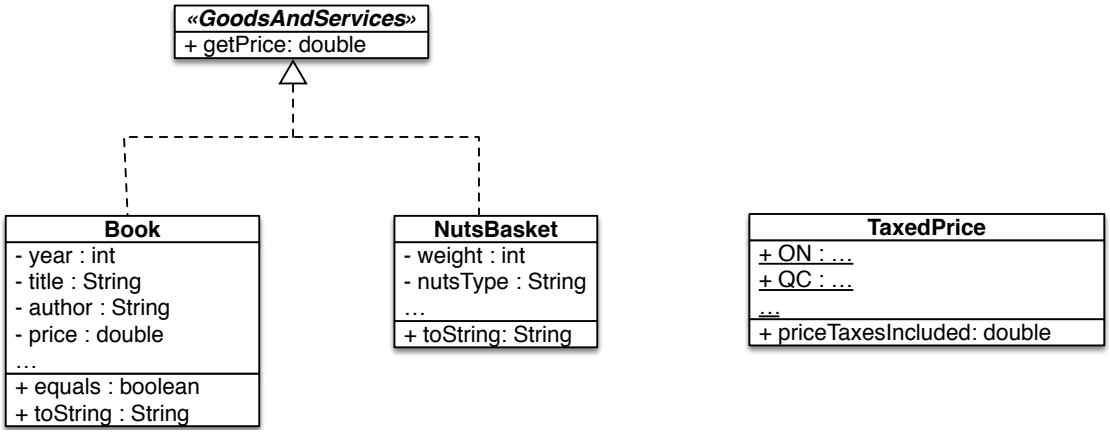
Barème

Question	Maximum	Résultat
1	35	
2	15	
3	15	
4	5	
Total	**70**	

Tous droits réservés. Il est interdit de reproduire ou de transmettre le contenu du présent document, sous quelque forme ou par quelque moyen que ce soit, enregistrement sur support magnétique, reproduction électronique, mécanique, photographique, ou autre, ou de l'emmagasiner dans un système de recouvrement, sans l'autorisation écrite préalable des instructeurs.

Question 1 : (35 points)

Cette question porte sur la programmation orientée objet en Java. Spécifiquement, il s'agit d'une petite portion d'un système informatique que vous développez pour une chaîne de supermarchés. Nous portons notre attention sur deux types de produits («*goods*»), des livres («*books*») et des paniers de noix («*baskets of nuts*»), ainsi que sur la partie du système informatique qui calcule le prix des produits et des services après taxes pour tous les items vendus par cette chaîne, dans chacune des provinces.



Dans cette application informatique, tout objet représentant un produit ou un service, incluant les objets des classes **Book** et **NutsBasket**, **peut être vu** («*can be seen*») comme **GoodsAndServices**.

- **GoodsAndServices** possède une méthode **getPrice**, retournant une valeur de type **double** qui représente le prix du produit ou du service **avant taxes**.
- **Book** possède quatre (4) variables d'instance : le prix du livre avant les taxes, de type **double**, l'année de publication, de type **int**, et deux variables d'instance de type **String**, l'une pour le titre et l'autre pour l'auteur du livre. Le **constructeur** de cette classe reçoit les valeurs initiales de ces variables en paramètre.

  Vous devez fournir les trois (3) méthodes suivantes pour la classe **Book** : premièrement, une méthode **toString**, dont le format est le même que celui de l'exemple ci-dessous. Deuxièmement, vous devez implémenter la méthode **equals**. On considère deux livres comme "égaux", s'ils ont le même titre et auteur, et ont été publiés la même année. Finalement, la méthode **getPrice** retourne le prix avant taxe de ce livre, cette valeur a été passée en paramètre au constructeur.
- **NutsBasket** possède 2 variables d'instance, **nutsType** de type **String**, et le poids («weight») en grammes, de type **int**. Le constructeur de la classe reçoit ces deux éléments d'information en paramètre. La classe doit aussi implémenter la méthode **toString**. Dans le cas de la classe **NutsBasket**, le prix avant taxe est de $0.1\$$ par gramme.

Le code source Java ci-dessous montre l'utilisation prévue des classes et de leurs méthodes.

```
Book b1, b2, b3;

b1 = new Book(1996, "John Smith", "Java for dummies", 85.0);
b2 = new Book(1996, "John Smith", "Java for dummies", 40.0);
b3 = new Book(2010, "Jane Adam", "Python for dummies", 50.0);

System.out.println(b1);
System.out.println(b2);
System.out.println(b3);
System.out.println(b1.equals(b2));
System.out.println(b1.equals(b3));

NutsBasket n1, n2;

n1 = new NutsBasket("Pistachios", 150);
n2 = new NutsBasket("Almonds", 400);

System.out.println(n1);
System.out.println(n2);
```

L'exécution du programme ci-dessus produira le résultat suivant sur la sortie.

```
"Java for dummies" by John Smith, sold for 85.0$ plus taxes
"Java for dummies" by John Smith, sold for 40.0$ plus taxes
"Python for dummies" by Jane Adam, sold for 50.0$ plus taxes
true
false
basket of Pistachios costing 15.0$ plus taxes
basket of Almonds costing 40.0$ plus taxes
```

Dans les boîtes prévues à cet effet sur les pages suivantes, vous devez fournir tout le code source nécessaire pour que le programme fonctionne tel que décrit.

Finalement, vous devez créer la classe **TaxedPrice**. On utilise sa méthode de classe **priceTaxesIncluded** pour calculer le prix d'un produit ou d'un service avec les taxes, pour une province donnée.

Pour l'instant, le système informatique ne traite que deux provinces. En Ontario, la taxe sur les produits et les services est de 13%. Alors qu'au Québec, la taxe sur les produits et les services est de 14,975%.

TaxedPrice possède une méthode de classe **priceTaxesIncluded** possédant deux paramètres, la référence d'un produit ou d'un service, et la province. Elle retourne le prix du produit ou du service incluant la taxe pour la province spécifiée en paramètre.

Le code source Java ci-dessous montre l'utilisation prévue de cette classe.

```
Book b1;

b1 = new Book(1996,"John Smith","Java for dummies", 85.0);

NutsBasket n1;

n1 = new NutsBasket("Pistachios", 150);

System.out.println("With taxes: " + b1 + " costs "
        +TaxedPrice.priceTaxesIncluded(b1,TaxedPrice.ON) + " in Ontario and "
        +TaxedPrice.priceTaxesIncluded(b1,TaxedPrice.QC) + " in Quebec.");

System.out.println("With taxes: " + n1 + " costs "
        +TaxedPrice.priceTaxesIncluded(n1,TaxedPrice.ON) + " in Ontario and "
        +TaxedPrice.priceTaxesIncluded(n1,TaxedPrice.QC) + " in Quebec.");
```

L'exécution du programme ci-dessus produira le résultat suivant sur la sortie.

```
With taxes: "Java for dummies" by John Smith, sold for 85.0$ plus taxes
costs 96.05 in Ontario and 97.72875 in Quebec.

With taxes: basket of Pistachios costing 15.0$ plus taxes costs 16.95 in
Ontario and 17.24625 in Quebec.
```

Dans la boîte prévue à cet effet, complétez l'implémentation de la classe **TaxedPrice**.

Question 2 : (15 points)

Avec le devoir 1, nous avons introduit le concept d'ajustement d'une courbe à l'aide de la régression linéaire. Le lissage («*smoothing*») est un concept similaire, qui trouve ses applications en statistiques et dans le traitement d'image. Alors que l'ajustement de courbe utilise une fonction explicite, le lissage modifie progressivement les valeurs d'un ensemble de données pour éliminer le bruit et approximer une fonction inconnue. Pour la classe **Smoothing** sur la page suivante, vous devez implémenter les méthodes **printArray** et **smooth** (lissage).

- La méthode **printArray** reçoit en paramètre la référence d'un tableau de valeurs de type **double**, et elle affiche les valeurs comme pour l'exemple ci-dessous.
- La méthode **smooth** reçoit elle aussi en paramètre la référence d'un tableau dont les valeurs sont de type **double**, et doit faire le «lissage» («*smoothing*») des valeurs. Cela signifie que chaque valeur est remplacée par la valeur moyenne de la valeur qui la précède, la valeur elle même, et la valeur qui la suit. La première valeur du tableau est remplacée par la moyenne de sa valeur et de celle qui la suit. La dernière valeur est remplacée par la moyenne des deux dernières valeurs du tableau. Par exemple, un tableau contenant les valeurs suivantes : $[2.0, 3.0, 1.0, 5.0]$ sera transformé comme suit : $[2.5, 2.0, 3.0, 3.0]$ par l'appel à la méthode **smooth**.

```
double[] test;

test = new double[]{0.0,5.0,0.0,5.0};
Smoothing.testSmooth(test);

test = new double[]{0.0,5.0,5.0,0.0};
Smoothing.testSmooth(test);

test = new double[]{0.0};
Smoothing.testSmooth(test);

test = new double[]{0.0,1.0};
Smoothing.testSmooth(test);

test = new double[]{2.0, 3.0, 1.0, 5.0};
Smoothing.testSmooth(test);

test=null;
Smoothing.testSmooth(test);
```

L'exécution du programme ci-dessus produira le résultat suivant sur la sortie.

```
Running testSmooth on: [0.0, 5.0, 0.0, 5.0]
After smoothing: [2.5, 1.6666666666666667, 3.3333333333333335, 2.5]
Running testSmooth on: [0.0, 5.0, 5.0, 0.0]
After smoothing: [2.5, 3.3333333333333335, 3.3333333333333335, 2.5]
Running testSmooth on: [0.0]
After smoothing: [0.0]
Running testSmooth on: [0.0, 1.0]
After smoothing: [0.5, 0.5]
Running testSmooth on: [2.0, 3.0, 1.0, 5.0]
After smoothing: [2.5, 2.0, 3.0, 3.0]
Not a valid value, null, for the method testSmooth.
```

Complétez l'implémentation de la classe **Smoothing**.

```
public class Smoothing {

    private static void testSmooth(double[] a) {
        if (a == null) {
            System.out.println("Not a valid value, null, for the method testSmooth.");
            return;
        }
        System.out.print("Running testSmooth on: ");
        printArray(a);
        smooth(a);
        System.out.print("After smoothing: ");
        printArray(a);
    }

    private static void printArray(double[] a) {









    }
```

```
    private static void smooth(double[] a) {

    }
}
```

Question 3 : (15 points)

Pour la classe **StackInspector** sur la page suivante, complétez l'implémentation de la méthode **findLast(E elem)**.

- La méthode **findLast** retourne l'index de la dernière occurrence de l'élément spécifié. S'il y a plus d'une occurrence, la dernière occurrence est celle qui se trouve le plus près du bas de la pile. C'est aussi celle avec l'index le plus élevé.
- La méthode retourne la valeur -1 si l'élément est absent de cette pile.
- La valeur de l'index augmente à partir du haut jusqu'au bas de la pile. L'index de l'élément du dessus est 0.
- La méthode **findLast** ne change pas l'état de la pile, c'est à dire que la pile contiendra les mêmes éléments dans le même ordre, avant et après un appel à la méthode.
- La valeur **null** n'est pas une valeur valide pour le paramètre **elem**.

L'exemple ci-dessous illustre le comportement voulu de la méthode **findLast**.

```
Stack<String> stack;
stack = new StackImplementation<String>();

stack.push("one");
stack.push("two");
stack.push("one");
stack.push("two");
stack.push("three");

StackInspector<String> inspector;
inspector = new StackInspector<String>(stack);

System.out.println(inspector.findLast("one"));
System.out.println(inspector.findLast("two"));
System.out.println(inspector.findLast("three"));
System.out.println(inspector.findLast("four"));
```

L'exécution du programme ci-dessus produira la sortie suivante sur la console.

```
4
3
0
-1
```

Pour cette question, il y a une classe nommée **StackImplementation** qui réalise l'interface **Stack**. Une instance de la classe **StackImplementation** peut sauvegarder un nombre arbitrairement grand d'éléments. Vous n'avez pas à fournir l'implémentation, elle vous est fournie.

```
public class StackInspector<E> {

    private Stack<E> stack;

    public StackInspector(Stack<E> stack) {
        this.stack = stack;
    }









}
```

```java
/**
 * Stack Abstract Data Type. A Stack is a linear data structure following
 * last-in-first-out protocol, i.e. the last element that has been added
 * onto the Stack, is the first one to be removed.
 */

public interface Stack<E> {

    /**
     * Tests if this Stack is empty.
     *
     * @return true if this Stack is empty; and false otherwise.
     */

    boolean isEmpty();

    /**
     * Returns a reference to the top element; does not change the state
     * of this Stack.
     *
     * @return The top element of this stack without removing it.
     */

    E peek();

    /**
     * Removes and returns the element at the top of this stack.
     *
     * @return The top element of this stack.
     */

    E pop();

    /**
     * Puts an element onto the top of this stack.
     *
     * @param element the element be put onto the top of this stack.
     */

    void push(E element);

}
```

Question 4 : (5 points)

Vous trouverez ci-dessous une expression en format **postfixe**. À l'aide de l'algorithme présenté en classe, vous devez transformer cette expression et donner l'expression **infixe** correspondante.

4 3 * 7 6 − * 2 4 + /