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 2019, 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 aucune page ou l'agrafe de cet examen.
7. Attendez l'annonce de début de l'examen.

Barème

Question	Maximum	Résultat
1	10	
2	5	
3	20	
4	15	
Total	**50**	

Consignes

- Pour l'ensemble des questions de cet examen, à l'exception des classes **System** et **Math**, vous ne devez pas utiliser les librairies de Java. Spécifiquement, vous ne devez pas utiliser les classes **Arrays** et **ArrayList**. Il ne doit y avoir aucun énoncé **import**.

Question 1 (10 points)

Pour cette question, vous devez implémenter la classe **Product** dont voici les caractéristiques.

- Possède une variable de classe **taxRate** (taux de taxation) de type **double**. Sa valeur initiale est **0.13**.
- Chaque objet de la classe **Product** possède une description (de type **String**) et un prix (**price** de type **double**). En conséquence, le constructeur de la classe possède deux paramètres correspondant à ces deux variables. Supposez que la valeur du paramètre **price** (prix) est positive.
- Il y a une méthode de classe **setTaxRate** pour l'affectation d'une nouvelle valeur à la variable **taxRate**. Supposez que la valeur du paramètre sera dans l'intervalle de 0.0 à 1.0.
- Il y a une méthode d'instance **getPriceWithTax** qui retourne le prix de ce produit, taxe incluse.
- La classe **Product** redéfinit la méthode **equals** de la classe **Object**. Assurez-vous que votre méthode est aussi robuste que possible.

Donnez votre implémentation de la classe **Product** dans l'espace prévu sur la page qui suit.

Implémentez la classe **Product** dans l'espace ci-dessous.

Question 2 (5 points)

Pour cette question, supposez l'existence des classes **QueueImplementation** et **StackImplementation**. Ces dernières réalisent les interfaces **Queue** et **Stack**, respectivement. Voici d'ailleurs les déclarations de ces interfaces.

```
public interface Queue<E> {

    /**
     * Retourne true si et seulement si cette file est vide.
     * @return true si la file est vide
     */

    boolean isEmpty();

    /**
     * Ajoute la référence de l'objet désigné par elem à l'arrière de la file.
     * @param elem la référence du nouvel élément
     */

    void enqueue(E elem);

    /**
     * Retire et retourne l'élément avant de cette file.
     * @return la référence de l'élément retiré
     */

    E dequeue();

}
```

```
public interface Stack<E> {

    /**
     * Retourne true si et seulement si cette pile est vide.
     * @return true si la pile est vide
     */

    boolean isEmpty();

    /**
     * Ajoute la référence de l'objet désigné par elem sur le dessus de la pile.
     * @param elem la référence du nouvel élément
     */

    void push(E elem);

    /**
     * Retire et retourne la référence de l'élément du dessus de la pile.
     * @return la référence de l'élément retiré
     */

    E pop();

}
```

Analysez attentivement le code source ci-dessous et donnez le résultat qui sera affiché sur la sortie lors de son exécution.

```
public class Test {

    public static void testQueue() {

        Queue<String> q;
        q = new QueueImplementation<String>();

        q.enqueue("");

        for (int i=0; i<7; i++) {
            String elem;
            elem = q.dequeue();
            System.out.println("["+elem+"]");
            q.enqueue(elem+"0");
            q.enqueue(elem+"1");
        }

    }

    public static void testStack() {

        Stack<String> s;
        s = new StackImplementation<String>();

        s.push("");

        for (int i=0; i<7; i++) {
            String elem;
            elem = s.pop();
            System.out.println("["+elem+"]");
            s.push(elem+"0");
            s.push(elem+"1");
        }

    }

    public static void main(String[] args) {
        System.out.println("Calling testQueue() ::");
        testQueue();
        System.out.println("Calling testStack() ::");
        testStack();
    }

}
```

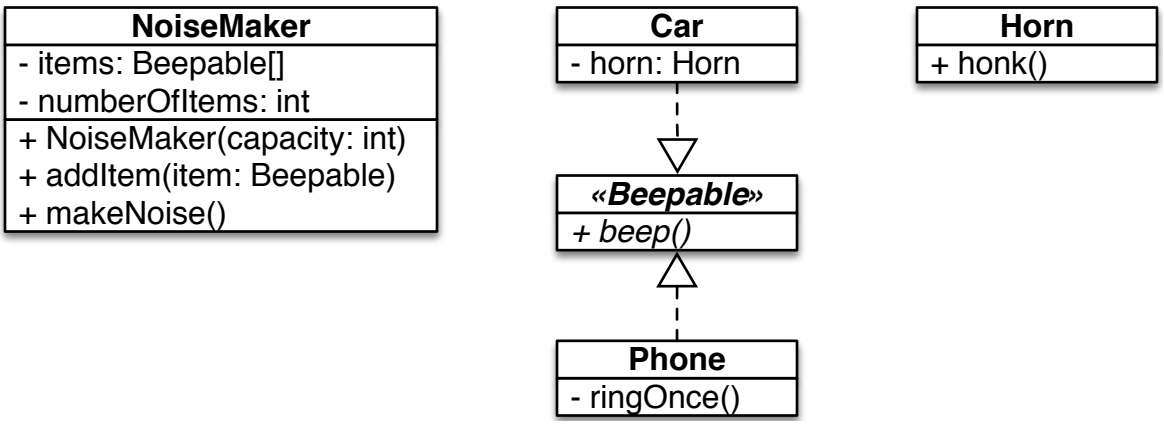
Donnez votre réponse dans l'espace prévu sur la page qui suit.

Donnez le résultat de l'exécution du programme **Test**.

```
> java Test
```

Question 3 (20 points)

Cette question comporte plusieurs classes en lien avec l'interface **Beepable**. Les diagrammes UML ci-dessous présentent leurs relations et leurs caractéristiques.



- Tous les objets qui réalisent l'interface **Beepable** ont une méthode **beep** (bip).
- Les classes **Car** (automobile) et **Phone** (téléphone) peuvent être vues («*can be seen as*») des objets **Beepable**.
- Un téléphone (objet de la classe **Phone**) fait «*beep*» (bip) à l'aide d'un appel à sa méthode **ringOnce()** (sonne une fois), qui affiche simplement la chaîne de caractères "ring !".
- Une automobile (objet de la classe **Car**) fait «*beep*» (bip) à l'aide d'un appel à la méthode **honk()** de son klaxon (avertisseur sonore - «*horn*»), qui affiche simplement la chaîne de caractères "honk !".
- Un objet **NoiseMaker** (faiseur de bruit) sauvegarde un maximum de **n** objets **Beepable**, où la valeur de **n** est passée en paramètre au constructeur. Supposez que la valeur de **n** sera toujours positive.
- La méthode **addItem** permet l'ajout d'un objet **Beepable** à cet objet **NoiseMaker**. Cette méthode affiche le message "This NoiseMaker is full" si le tableau est plein et ignore cet item. De plus, la méthode affiche le message "null is not a valid value" et ignore cet item, si la valeur du paramètre est **null**.
- Lorsque la méthode **makeNoise** (faire du bruit) est appelée, l'objet doit demander aux objets sauvegardés (les **Beepable**) de faire bip («*beep*»).

En particulier, l'exécution des énoncés suivants :

```
NoiseMaker m;
m = new NoiseMaker(5);
m.addItem(new Phone());
m.addItem(new Car());
m.addItem(new Car());
m.addItem(new Phone());
m.addItem(new Phone());
m.addItem(new Car());
m.makeNoise();
```

doit produire cette sortie :

```
This NoiseMaker is full
ring!
honk!
honk!
ring!
ring!
```

A. Implémentez l'interface **Beepable**.

B. Implémentez la classe **Car**.

Voici la classe **Horn**.

```
public class Horn {

    public void honk() {
        System.out.println("honk!");
    }

}
```

C. Implémentez la classe **Phone**.

D. Implémentez la classe **NoiseMaker**.

Question 4 (15 points)

Pour cette question, vous donnez l'implémentation d'une classe afin de représenter un polynôme («*polynomial*» en anglais). Un polynôme est une formule ayant cette forme $f(t) = 2.0 + 4.0 \times t^2 - t^3$. Ce polynôme-ci a un degré 3 et les coefficients suivants : 2.0, 0.0, 4.0, et -1.0.

- Spécifiquement, vous devez sauvegarder les coefficients du polynôme dans un tableau.
- Il y a deux constructeurs. L'un d'eux reçoit la référence d'un tableau qui contient les coefficients à utiliser pour l'initialisation de ce polynôme. Supposez que cette référence n'est pas **null**. Le second constructeur reçoit le degré de ce polynôme seulement. Supposez que le degré est positif. Pour ce second constructeur, tous les coefficients ont la valeur 0.0 au moment de l'initialisation.
- Il y a une méthode **set(int index, double value)** qui permet de changer la valeur du coefficient à l'index spécifié. Supposez que la valeur de l'index est valide pour ce polynôme.
- La méthode **get(int index)** retourne la valeur du coefficient à l'index spécifié. Supposez que la valeur de l'**index** est valide pour ce polynôme.
- La méthode **eval(double t)** calcule la valeur du polynôme pour la valeur de **t** spécifiée. L'évaluation du polynôme ci-dessus pour la valeur 2.0 retourne 10.0, c'est-à-dire $2.0 + 4.0 \times 2.0^2 - 2.0^3$. Suggestion : utilisez la méthode **Math.pow(base,exposant)** pour le calcul.
- La méthode **toString** retourne une chaîne de caractères qui représente ce polynôme. Le format attendu est présenté sur la page qui suit.

Assurez-vous que l'exécution du programme ci-dessous à l'aide de votre implémentation produira le résultat attendu.

```
public class TestPolynomial {

    public static void main(String[] args) {

        Polynomial f,g,h;

        double[] coefficients;
        coefficients = new double[]{2.0, 0.0, 4.0, -1.0};

        f = new Polynomial(coefficients);

        coefficients[1] = 3.0;

        g = new Polynomial(coefficients);

        h = new Polynomial(12);

        h.set(0,  7.0);
        h.set(2,  4.0);
        h.set(6, -2.0);
        h.set(12,  5.0);

        System.out.println(f);
        System.out.println(g);
        System.out.println(h);

        System.out.println(f.eval(2));

    }

}
```

Résultat attendu :

```
2.0 + 4.0 * t^2 + (-1.0) * t^3
2.0 + 3.0 * t^1 + 4.0 * t^2 + (-1.0) * t^3
7.0 + 4.0 * t^2 + (-2.0) * t^6 + 5.0 * t^12
10.0
```

```
public class Polynomial {

    // Variable(s) d'instance




    // Constructeurs
    public Polynomial(double[] coefficients) {









    }
    public Polynomial(int degree) {









    }
    // Setter
    public void set(int index, double value) {


    }
    // Getter
    public double get(int index) {


    }
```

```
    // Méthodes d'instance

    public double eval(double t) {









    }

    public String toString() {

```

```
    }
}
```