Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Introduction à l'informatique II (ITI 1521)
Examen final

Instructeur: Miguel Garzón

Avril 2013, durée: 3 heures

Identification

Nom : _____ Prénom : _____

d'étudiant : _____ # place : _____ Signature : _____

Consignes

1. **Lisez ces consignes ;**
2. Livres fermés ; Sans calculatrice ou toute autre forme d'aide ;
3. Les appareils électroniques, ou tout autre dispositif de communication, ne sont pas autorisés ; Tout appareil doit être éteint, rangé et hors de portée ; Quiconque contrevient au présent règlement peut être accusé de fraude scolaire ;
4. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle ;
5. Écrivez lisiblement, votre note en dépend ; Commentez vos réponses ; Ne retirez pas l'agrafe.

Barème

Question	Maximum	Résultat
1	10	
2	10	
3	25	
4	15	
5	20	
6	10	
7	10	
Total	**100**	

Question 1 (10 points)

A. La valeur d'une variable d'un type primitif se trouve à l'adresse désignée par l'étiquette (identificateur).
Vrai ou **faux**.

B. Le constructeur d'un objet porte le même nom que la classe et possède une valeur de retour.
Vrai ou **faux**.

C. Une classe abstraite contient uniquement des méthodes abstraites.
Vrai ou **faux**.

D. La redéfinition d'une méthode dans une sous-classe cache la méthode d'origine de la super classe.
Vrai ou **faux**.

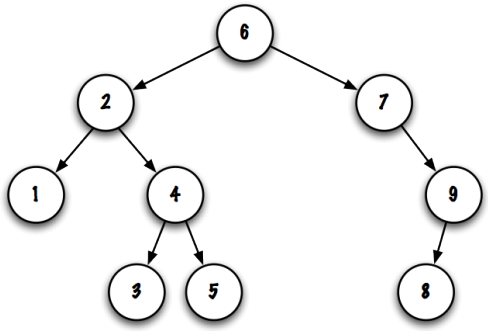
E. Une classe final est une classe qui ne peut pas avoir de sous-classes.
Vrai ou **faux**.

F. La valeur de l'expression postfixe :

```
20 35 - 5 / 10 7 * +
```

est 67.
Vrai ou **faux**.

G. Lorsqu'on traverse l'arbre binaire de recherche ci-dessous afin de visiter tous ses nœuds en utilisant un parcours postfixe, le rendu du parcours est : 1, 3, 5, 2, 4, 8, 9, 7, 6.



Vrai ou **faux**.

H. Les méthodes récursives auront toujours au moins un paramètre de plus que leur contrepartie publique.

Vrai ou **faux**.

I. Une méthode qui lance des exceptions de type "unchecked" doit les déclarer en utilisant le mot-clé **throws**.
Vrai ou **faux**.

J. Une classe imbriquée non-statique peut avoir accès aux méthodes et aux attributs de la classe englobante (outer) même si ces derniers sont déclarés privés.
Vrai ou **faux**.

Question 2 (10 points)

A. Quel type abstrait de données l'algorithme d' **analyse syntaxique** utilise-t-il ?

(a) Queue

(b) List

(c) BinarySearchTree

(d) aucune de ces réponses

Réponse :

B. Quel énoncé décrit le mieux le bloc de code qui suit ? Encerclez votre choix.

(a) Affiche "c = 0" ;

(b) Affiche "c = Infinity" ;

(c) Affiche "** caught Exception **", "c = 2" ;

(d) Affiche "** caught ArithmeticException **", "c = 3" ;

(e) Produira une erreur d'exécution, ainsi qu'une trace d'exécution.

(f) Erreur de compilation : "exception java.lang.ArithmeticException has already been caught" ;

```
int a = 1, b = 0, c = 0;
try {
    c = a/b;
} catch ( Exception e ) {
    System.err.println( "** caught Exception **" );
    c = 2;
} catch ( ArithmeticException ae ) {
    System.err.println( "** caught ArithmeticException **" );
    c = 3;
}
System.out.println( "c = " + c );
```

Réponse :

C. Au sujet de l'implémentation **ArrayList** de l'interface **List**.

(a) Les insertions aux positions intermédiaires sont toujours rapides

(b) L'ajout d'un élément à la première position est toujours rapide

(c) Le retrait du premier élément est toujours rapide

(d) Les accès directs en lecture aux positions intermédiaires sont toujours rapides

Réponse :

D. Au sujet de l'implémentation **LinkedList** de l'interface **List**. Afin d'accélérer l'ajout à l'arrière d'une liste simplement chaînée :

(a) On peut ajouter les éléments dans l'ordre inverse.

(b) On peut ajouter une nouvelle variable d'instance pointant sur le dernier élément de la liste.

(c) On peut ajouter une nouvelle variable d'instance 'previous' qui pointe sur l'avant dernier élément de la liste.

(d) (b) et (c)

(e) aucune de ces réponses

Réponse :

E. Considérez l'implémentation de la classe **CircularQueue** ci-bas. Étant donné une file **q** contenant les éléments suivants : "A,B,C,D,E,F,G", où **A** est l'élément avant de la file, à la suite de l'appel suivant **q.magic(4)**, le contenu de la file **q** sera :

(a) "E,F,G"

(b) "A,B,C,D"

(c) "E,F,G,A,B,C,D"

(d) "E,F,G,A,B,C,D,A,B,C,D"

(e) aucune de ces réponses

```
public class CircularQueue<E> {
    private E[] elems;
    private int front;
    private int rear;

    public CircularQueue(int capacity) {
        if (capacity < 0) {
            throw new IllegalArgumentException("negative number");
        }
        elems = (E[]) new Object[capacity];
        front = -1;
        rear = -1;
    }

    public void magic(int n) {
        if (rear != -1 && rear != front) {
            while (n > 0) {
                E current = elems[front];
                elems[front] = null;
                front = (front + 1) % elems.length;
                rear = (rear + 1) % elems.length;
                elems[rear] = current;
                n--;
            }
        }
    }
}
```

Question 3 (25 points)

Vous devez implémenter un **Système de gestion de lettres de références** pour l'Université d'Ottawa. À l'Université d'Ottawa les étudiants potentiels qui appliquent aux programmes de maîtrise et de doctorat doivent fournir des lettres de recommandation d'anciens professeurs ou employeurs.

Vous devez concevoir en Java l'implémentation des classes **Person**, **Student**, **Reference** et **Letter**. Ajoutez tous les constructeurs nécessaires. Chaque attribut doit avoir un accesseur (getter). Voici un programme test pour illustrer l'usage des classes de cette application.

```
Person student, reference1, reference2;

student = new Student("Lionel ", "cr7@uottawa.ca", "incognito", "432534", 9.2);
reference1 = new Reference("Pepe", "pepe@uottawa.ca", "wrestling","IEEE");
reference2 = new Reference("Mourinho", "specialone@uottawa.ca","psgmission", "SITE");

Letter letter1 = new Letter("Student is excellent ", 5, 5, 5);
Letter letter2 = new Letter("A hard worker student, very innovative", 5, 3, 4);

student.add(letter1);
student.add(letter2);

reference1.add(letter1);
reference2.add(letter2);

for (Letter aLetter : student.getLetters()) {
    if (aLetter != null) {
        System.out.println(aLetter);
    }
}
```

L'exécution du programme Java ci-dessus produira le résultat qui suit en sortie :

```
Comment: Student is excellent; Performance = 5.0; Originality = 5.0;
  Potential = 5.0.
Comment: A hard worker student, very innovative; Performance = 5.0; Originality = 3.0;
  Potential = 4.0.
```

Pour cette question, vous êtes autorisé à utiliser la classe prédéfinie **java.util.ArrayList**. En particulier, cette classe réalise l'interface **List**, possède un constructeur **ArrayList()** et peut stocker un grand nombre d'éléments. En plus, elle implémente les méthodes suivantes :

```
public interface List<E> {
    // Appends the specified element to the end of this list
    public abstract boolean add(E o);
    // Returns the element at the specified position in this list.
    public abstract E get(int index);
    // Replaces the element at the specified position with the specified element
    public abstract E set(int index, E element);
    // Returns an array containing all of the elements in this list in proper order
    public abstract Object[] toArray();
    // Returns the number of elements in this list
    public abstract int size();
}
```

A. La classe **Person** possède des champs afin de sauvegarder le nom de la personne, son adresse courriel, et un mot de passe. Une personne peut avoir plusieurs lettres de recommandation. Assurez-vous d'y inclure au moins un constructeur, ainsi que les méthodes d'accès appropriées.

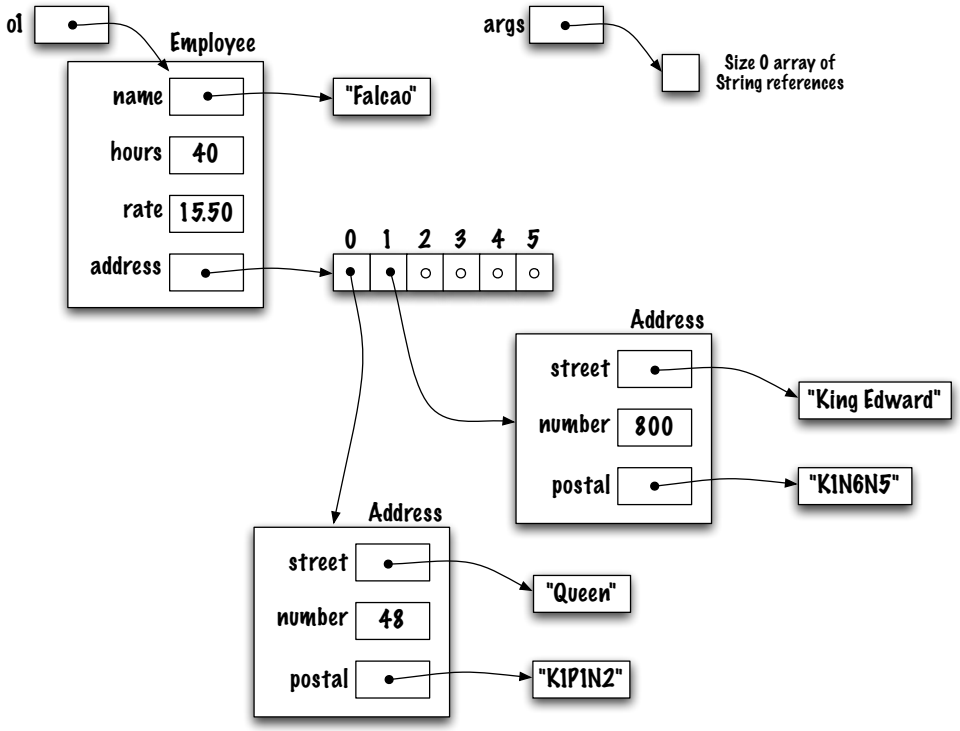
B. Un étudiant **Student** est une personne (Person) qui dispose également d'un numéro d'étudiant et d'une moyenne d'admission. Un étudiant peut avoir plusieurs lettres de recommandation. Assurez-vous d'y inclure au moins un constructeur, ainsi que les méthodes d'accès appropriées.

C. Un recommandataire **Reference** est un personne qui possède un champ pour garder le nom de l'institution à laquelle il appartient. Un recommandataire peut écrire beaucoup de lettres (une par étudiant qui la lui a demandée). Assurez-vous d'y inclure au moins un constructeur, ainsi que les méthodes d'accès appropriées.

D. Une lettre de recommandation **Letter** possède un champ pour sauvegarder des commentaires généraux sur l'étudiant, les évaluations (de 0(faible) à 5 (élevé)) pour la performance, l'originalité et le potentiel de recherche.

Question 4 (15 points)

A. En vous basant sur le diagramme de mémoire ci-dessous, écrivez l'implémentation de toutes les classes, variables d'instances et constructeurs.




```
// End of Question
```

Hint : Vous avez besoin d'une classe **Employee** (avec une méthode **main**) et d'une classe **Address**.

// End of Question

Question 5 (20 points)

Le type abstrait de données (TAD) **Deque** ("Double-Ended QUEue") combine à la fois caractéristiques d'une file et d'une pile. En particulier, le TAD **Deque** - prononcé "deck" - permet :

- des insertions efficaces à l'avant et à l'arrière de la structure de données ;
- des retraits efficaces à l'avant et à l'arrière de la structure de données.

Voici une description des quatre méthodes d'accès de cette classe.

- **void offerFirst(E item)** : ajoute un item à l'avant de ce **Deque** ;
- **void offerLast(E item)** : ajoute un item à l'arrière de ce **Deque** ;
- **E pollFirst()** : retire et retourne l'item à l'avant de ce **Deque**, retourne **null** si ce **Deque** était vide ;
- **E pollLast()** : retire et retourne l'item à l'arrière de ce **Deque**, retourne **null** si ce **Deque** était vide.

Vous trouverez ci-dessous une implémentation partielle de la classe **LinkedDeque** qui utilise des éléments chaînés pour la sauvegarde des éléments. Complétez l'implémentation de la classe **LinkedDeque**.

- Les éléments sont doublement chaînés ;
- Un objet **LinkedDeck** possède deux variables d'instance, **front** et **rear**.

```
public class LinkedDeque<E> implements Deque<E> {

    private class Node<T> {

        private T value;
        private Node<T> prev;
        private Node<T> next;

        private Node(T value, Node<T> prev, Node<T> next) {
            this.value = value;
            this.prev = prev;
            this.next = next;
        }
    }

    LinkedDeque() {
        front = null;
        rear = null;
    }

    private Node<E> front;
    private Node<E> rear;
```

Hint : Dessinez les diagrammes mémoire. Considérez tous les cas possibles.

```
    public void offerFirst(E e) {

    } // End of offerFirst
```

```
    public void offerLast(E e) {

    } // End of offerLast
```

```
    public E pollFirst() {









    } // End of pollFirst
```

```
    public E pollLast() {









    } // End of pollLast
} // End of LinkedDeque
```

Question 6 (10 points)

Pour la classe **LinkedList** ci-dessous, écrivez l'implémentation de la méthode **LinkedList<E> partition(E elem)**. Cette méthode d'instance partitionne la liste en deux parties. Cette instance conserve les éléments les plus à gauche jusqu'à la première occurrence de **elem**, incluant ce dernier. Le reste des éléments sont retournés dans une nouvelle liste. Si **elem** n'est pas trouvé dans la liste, alors la liste reste intacte (aucun changement est effectué) et la liste retournée est vide.

Exemple : si **xs** désigne une liste contenant les éléments suivants **[1,2,3,4,5,6]**,suite à l'appel de méthode **ys = xs.partition(3)**, la liste désignée par **xs** contient maintenant les éléments suivants **1,2,3**, et **ys** désigne une liste contenant ces éléments **4,5,6**.

Vous devez implémenter la méthode à l'aide de la technique présentée en classe pour l'implémentation de méthodes récursives à l'intérieur de la classe. Il y a donc une partie publique qui sert à lancer le premier appel à la méthode privée, récursive.

```
public class LinkedList<E> {

    private static class Node<E> {

        private E value;
        private Node<E> next;

        private Node(E value, Node<E> next) {
            this.value = value;
            this.next = next;
        }
    }

    private Node<E> head = null;
```

```
    public LinkedList<E> partition(E elem) {









    } // End of partition
```

```
    private LinkedList<E> partitionRec(                                    ) {









    } // End of partitionRec
} // End of LinkedList
```

Question 7 (10 points)

Pour la classe **BinarySearchTree** ci-dessous, vous devez implémenter la méthode d'instance **int countIf(E elem)**. Cette méthode retourne le nombre d'éléments se trouvant dans l'arbre qui sont supérieurs à **elem**.

- Les éléments sauvegardés dans l'arbre binaire de recherche réalisent l'interface **Comparable<E>**. Rappelez vous que la méthode **int compareTo(E other)** retourne un nombre négatif, zéro, ou un entier positif si l'instance est plus petite, égale à, ou supérieure à l'objet désigné par **other**.
- Une méthode qui visite trop de nœuds obtiendra une note maximale de 9 points.
- **Exemple** si **t** désigne un arbre binaire de recherche contenant les éléments suivants **1,2,3,4,5,6**, l'appel de méthode **t.countIf(2)** retournera **4**.

```
public class BinarySearchTree<E extends Comparable<E> > {

    private static class Node<E> {

        private E value;

        private Node<E> left;
        private Node<E> right;

        private Node( E value ) {
            this.value = value;
            left = null;
            right = null;
        }
    }

    private Node<E> root = null;
```

```
} // End of BinarySearchTree
```

(blank space)

(blank space)