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 final

Instructeur: Marcel Turcotte

Avril 2010, durée : 3 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	10	
2	10	
3	15	
4	15	
5	12	
6	13	
7	25	
Total	**100**	

Question 1 : (10 points)

Pour chaque énoncé, vous devez encercler la bonne réponse, soit vrai ou faux.

A. En Java, `total`, `ToTal` et `TOTAL` sont des identificateurs distincts.
Réponse : vrai faux

B. Une classe peut implémenter plusieurs interfaces.
Réponse : vrai faux

C. En Java, une classe n'a qu'un seul parent direct.
Réponse : vrai faux

D. Une sous-classe peut redéfinir une méthode de la classe parent ayant la même signature, c'est-à-dire ayant le même nom et les mêmes paramètres.
Réponse : vrai faux

E. Une classe abstraite doit contenir des méthodes abstraites.
Réponse : vrai faux

F. Une exception dite « *unchecked* » doit être traitée ou déclarée, sinon la compilation du programme va échouer.
Réponse : vrai faux

G. L'algorithme de fouille en largeur utilise une pile.
Réponse : vrai faux

H. On peut implémenter une pile ou une file de sorte que l'exécution des opérations nécessite un nombre constant d'instructions.
Réponse : vrai faux

I. Un programme causant un nombre infini d'appels récursifs se comporte comme une boucle infinie.
Réponse : vrai faux

J. Dans un arbre, un noeud n'ayant aucun descendant est une feuille.
Réponse : vrai faux

Question 2 : (10 points)

A. L'environnement de programmation « *Java Standard Edition Development Kit* » comprend un programme permettant l'archivage de plusieurs fichiers Java dans un seul fichier. Comment se nomme ce programme ?

(a) zip

(b) tar

(c) compress

(d) jar

(e) WinZip

Réponse :

B. Quels énoncés ou déclarations **ne sont pas** valides ?

(a) `next() = next + 1;`

(b) `a = b / c % d;`

(c) `int money, dollars = 0, cents = 0;`

(d) `float pi = 3.1415F;`

(e) all of the above

Réponse :

C. Ce programme Java :

(a) causera une erreur de compilation

(b) causera une erreur d'exécution

(c) affiche **true**

(d) affiche **false**

(e) affiche **"5 == 5"**

Réponse :

```
public class Test {
    public static void incr( int value ) {
        value = value + 1;
    }
    public static void incr( Integer value ) {
        value = value + 1;
    }
    public static void main( String[] args ) {
        int i1;
        Integer i2;
        i1 = 5;
        incr( i1 );
        i2 = 4;
        incr( i2 );
        System.out.println( i1 == i2 );
    }
}
```

Question 2: (suite)

D. Une classe dont la déclaration est préfixée par le mot-clé **final** :

(a) ne peut changer

(b) ne peut pas avoir de sous-classes

(c) ne peut pas avoir de superclasses

(d) possède plusieurs méthodes abstraites

(e) n'est pas valide en Java

Réponse :

E. Pour invoquer le constructeur de la super-classe, la sous-classe doit utiliser ce mot réservé ?

(a) abstract

(b) construct

(c) parent

(d) super

(e) extends

Réponse :

F. Quel mot-clé sert à définir une variable partagée par toutes les instances d'une classe ?

(a) static

(b) final

(c) public

(d) private

(e) aucune de ces réponses

Réponse :

G. Quel concept désigne un ensemble de valeurs et les opérations permises pour ces valeurs ?

(a) type abstrait de données

(b) générique

(c) collection

(d) *bag*

(e) aucune de ces réponses

Réponse :

Question 2: (suite)

H. Quel énoncé est un avantage des implémentations à base de tableaux par rapport aux implémentations à l'aide d'éléments chaînées ?

(a) la taille de la structure de données augmente et diminue au besoin

(b) la structure utilisée est de taille fixe

(c) on accède directement aux éléments

(d) toutes ces réponses

(e) ni (a), ni (b), ni (c)

Réponse :

I. Dans un arbre binaire de recherche, toutes les valeurs du sous-arbre droit de la racine sont :

(a) plus grandes que la valeur sauvegardée à la racine

(b) plus petites que la valeur sauvegardée à la racine

(c) plus grandes ou égales à la valeur sauvegardée à la racine

(d) plus petites ou égales à la valeur sauvegardée à la racine

(e) égales à la valeur sauvegardée à la racine

Réponse :

J. Lorsque l'élément retiré d'un arbre binaire de recherche est une feuille, pour s'assurer que l'arbre demeure un arbre binaire de recherche, il faut :

(a) remplacer cet élément par son fils

(b) remplacer cet élément par son successeur en ordre symétrique

(c) simplement retirer cet élément

(d) toutes ces réponses

(e) ni (a), ni (b), ni (c)

Réponse :

Question 3 : (15 points)

A. Donnez l'ordre des éléments de la pile suite à l'exécution de ces énoncés. Assurez-vous de clairement identifier l'élément du dessus.

```
s.push( new Integer( 8 ) );
s.push( new Integer( 6 ) );
Integer num = s.pop();
s.push( new Integer( 3 ) );
s.push( new Integer( 4 ) );
s.push( new Integer( 15 ) );
s.push( new Integer( 12 ) );
s.pop();
s.pop();
s.pop();
s.push( new Integer( 19 ) );
```

Réponse:

B. Dessinez l'arbre binaire de recherche qui sera produit par l'ajout des éléments suivants, dans cet ordre, si l'on utilise la méthode **add** présentée en classe : 12, 16, 9, 1, 15, 13.

Réponse:

C. Donnez le résultat de ce programme.

```
public class Test {

    public static int bar( int i ) {
        if ( i == 2 ) {
            throw new Exception( "oups!" );
        }
        return i;
    }

    public static int foo() {
        int result;
        result = bar( 1 );
        try {
            result = bar( 2 );
        } catch ( Exception e ) {
            result = bar( 3 );
        }
        return result;
    }

    public static void main( String[] args ) {
        System.out.println( foo() );
    }

}
```

Réponse:

D. La classe **Sequence** est une liste chaînée qui possède des méthodes permettant l'écriture, à l'extérieur de la classe, de méthodes pour le traitement récursif des listes. Voici les caractéristiques de la classe **Sequence**.

- **boolean isEmpty()** ; retourne **true** si et seulement si cette séquence est vide ;
- **Sequence<E> split()** ; retourne un objet qui contient le reste de cette séquence, l'instance ne contient alors qu'un élément. La méthode lance l'exception **IllegalStateException** si cette séquence était vide lors de l'appel ;
- **void join(Sequence<E> other)** ; ajoute les éléments de **other** à la suite des éléments de cette séquence, **other** est alors vide ;
- **deleteFirst()** retire le premier élément de cette **Sequence** .

```
public static <E> int april2010( Sequence<E> l ) {

    int result = 0;

    if ( ! l.isEmpty() ) {

        Sequence<E> tail = l.split();

        result = 1 + april2010( tail );

        if ( result % 2 == 0 ) {
            l.deleteFirst();
            result = 0;
        }

        l.join( tail );
    }

    return result;
}
```

Soit **s**, une séquence contenant les éléments qui suivent, dans cet ordre, {0,1,2,3,4}, quel sera le contenu de **s** à la suite de cet appel de méthode **april2010(s)** ?

Réponse:

Question 4 : (15 points)

À l'intérieur de la classe **SinglyLinkedList** ci-dessous, vous devez ajouter une méthode d'instance récursive qui retourne la liste des positions auxquelles se trouve l'objet passé en paramètre. Vous devez utiliser la technique présentée en classe. Ainsi, il y aura une méthode publique qui lance le premier appel à la méthode récursive privée.

Si **l** désigne une liste contenant les éléments {A, B, A, A, C, D, A}, alors l'appel **l.indexOfAll(A)** retourne la liste de positions qui suit {0, 2, 3, 6}.

```
public class SinglyLinkedList<E> {

    private static class Node<F> {

        private F value;
        private Node<F> next;

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

    // Variable d'instance
    private Node<E> first;

    // Méthodes d'instance
    public void addFirst( E item ) { ... }
    public void addLast( E item ) { ... }
```

```
    // Complétez l'implémentation de cette méthode

    public                              indexOfAll( E elem ) {









    } // Fin de indexOfAll
```

```
  // Complétez l'implémentation de cette méthode

  private                       indexOfAllRec(                                ) {









  } // Fin de indexOfAllRec
```

```
} // Fin de SinglyLinkedList
```

Question 5 : (12 points)

Pour l'implémentation partielle de la classe **CircularQueue** ci-bas, vous devez compléter l'implémentation de la méthode d'instance **void rotate(int n)**.

- La méthode **rotate(int n)** retire **n** éléments situés à l'avant de la file afin de les ajouter à l'arrière de la file. Les éléments seront ajoutés à l'arrière dans le même ordre qu'on les retire. Par exemple, étant donné une file **q** contenant les éléments suivants : « A, B, C, D, E», ou **A** est l'élément avant de la file, à la suite de l'appel suivant **q.rotate(2)**, le contenu de la file **q** sera «C, D, E, A, B» ;
- Le tableau **elems** sert à sauvegarder les éléments de la file ;
- **CircularQueue** utilise un tableau circulaire tel que vu en classe, ainsi, les valeurs des index avant et arrière font un bouclage automatique lorsqu'elles atteignent la limite du tableau ;
- La variable d'instance **front** désigne la cellule du tableau contenant l'élément avant de la file ou -1 si la file est vide ;
- La variable d'instance **rear** désigne la cellule du tableau contenant l'élément arrière de la file ou -1 si la file est vide ;
- Si la valeur du paramètre est négative, la méthode doit lancer l'exception **IllegalArgumentException** ;
- Pour cette implémentation de la méthode **rotate**, **vous ne devez pas utiliser les méthodes d'instance de cette classe**, en particulier, vous ne devez pas utiliser les méthodes **enqueue** et **dequeue**. Votre implémentation doit faire explicitement référence aux variables **front** et **rear**.

Complétez l'implémentation de la méthode **rotate(int n)** sur la page qui suit.

```
public class CircularQueue<E> implements Queue<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 rotate( int n ) {

    }

} // Fin de CircularQueue
```

Question 6 : (13 points)

Implémentez la méthode **rotate** de la question **5** mais cette fois-ci pour l'implémentation partielle de la classe **LinkedQueue** ci-dessous.

- Les éléments de la file sont sauvegardés dans une structure de données simplement chaînée ;
- La variable d'instance **front** désigne le premier élément de la file ou a la valeur **null** si la file est vide ;
- La variable d'instance **rear** désigne le dermier élément de la file ou a la valeur **null** si la file est vide ;
- Lorsque la valeur du paramètre est négative, la méthode doit lancer l'exception **IllegalArgumentException** ;
- Pour cette implémentation de la méthode **rotate**, **vous ne devez pas utiliser les méthodes d'instance de cette classe**, en particulier, vous devez transformer la structure de la liste en changeant les liens de la liste (les références **next**).

Complétez l'implémentation de la méthode **rotate(int n)** sur la page qui suit.

```
public class LinkedQueue<E> implements Queue<E> {

    private static class Elem<F> {

        private F value;
        private Elem<F> next;

        private Elem( F value, Elem<F> next ) {
            this.value = value;
            this.next = next;
        }
    }

    private Elem<E> front;
    private Elem<E> rear;

    public LinkedQueue() {
        front = rear = null;
    }
```

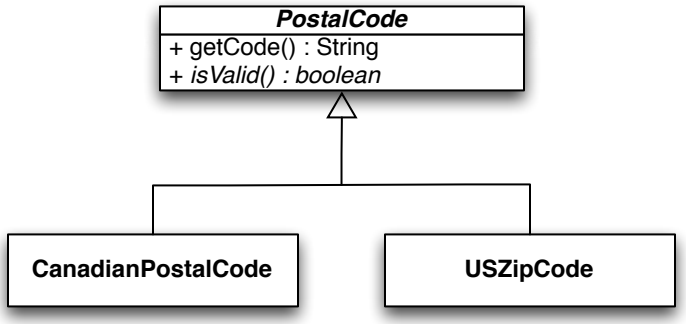
```
    public void rotate( int n ) {

    }

} // Fin de LinkedQueue
```

Question 7 : (25 points)

Le diagramme UML ci-bas présente une hiérarchie de classes afin de représenter des codes postaux de divers pays.



Sachant que,

- Tous les codes postaux ont une méthode **getCode** retournant le code (de type **String**) représenté par cette instance ;
- Tous les codes postaux ont une méthode **isValid** retournant **true** si le code de cette instance est valide, et **false** sinon ;
- Un code postal canadien est valide si les positions 0, 2 et 5 sont occupées par des lettres, les positions 1, 4 et 6 sont des chiffres, et la position 3 est un caractère blanc ;
- Un code postal américain (Zip code) valide est consistitué de deux lettres, suivies d'un espace blanc, suivi de 5 chiffres.

Concevoir une implémentation pour les classes **PostalCode**, **CanadianPostalCode** et **USZipCode**. Assurez-vous d'y inclure les variables d'instance, ainsi que les constructeurs. L'appendice A présente un résumé des méthodes des classes **String** et **Character**.

```
// Implémentez la classe PostalCode ici
```

Question 7: (suite)

```
// Implémentez la classe CanadianPostalCode ici
```

Question 7: (suite)

```
// Implémentez la classe USZipCode ici
```

A Appendice

La classe **String** comprend les méthodes suivantes.

char charAt(int pos) : retourne le caractère se trouvant à la position spécifiée par l'index **pos** ;

int length() : retourne la longueur de cette chaîne.

La classe **Character** comprend les méthodes suivantes.

static boolean isDigit(char ch) : détermine si le caractère passé en paramètre est un nombre ;

static boolean isLetter(char ch) : détermine si le caractère passé en paramètre est une lettre ;

static boolean isWhitespace(char ch) : détermine si le caractère passé en paramètre est un espace blanc [1].

1. On devrait dire « une espace » ; qui signifie « caractère électronique jouant le rôle de la tige métallique insérée entre les caractères de plomb et servant à créer des espaces entre les mots à imprimer. »