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 2011, durée : 2 heures

Identification

Nom, prénom : ______________________________

Numéro d'étudiant : ______________ Signature : ______________________

Consignes

1. Examen à 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	15	
2	32	
3	23	
Total	**70**	

Question 1 : (15 points)

Pour cette question, n articles sont sauvegardés dans k boîtes. Étant donné le poids des articles (**weights**) et leur affectation aux boîtes (**map**), vous devez implémenter en Java une méthode de classe calculant la différence de poids entre la boîte la plus lourde et celle la plus légère.

- Il y n articles, numérotés $0, 1 \ldots n-1$.
- Il y k boîtes, numérotées $0, 1 \ldots k-1$.
- Le paramètre formel **weights** désigne un tableau d'entiers de taille n tel que **weights[j]** est le poids du j-ième article. Ainsi, **weights[0]** est le poids de l'article 0, **weights[1]** est le poids de l'article 1, etc.
- Le paramètre formel **map** désigne un tableau d'entiers de taille n tel que **map[j]** indique le numéro de boîte où l'article **j** est sauvegardé. Ainsi, **map[0]** indique le numéro de boîte où l'article 0 est sauvegardé, **map[1]** indique le numéro de boîte où l'article 1 est sauvegardé, etc.

```java
public static int getWeightDifference( int[] weights, int[] map, int k ) {

}
```

En supposant que la méthode **getWeightDifference** est déclarée dans une classe nommée **Q1**, l'appel à la méthode **println** ci-dessous affichera la valeur 70.

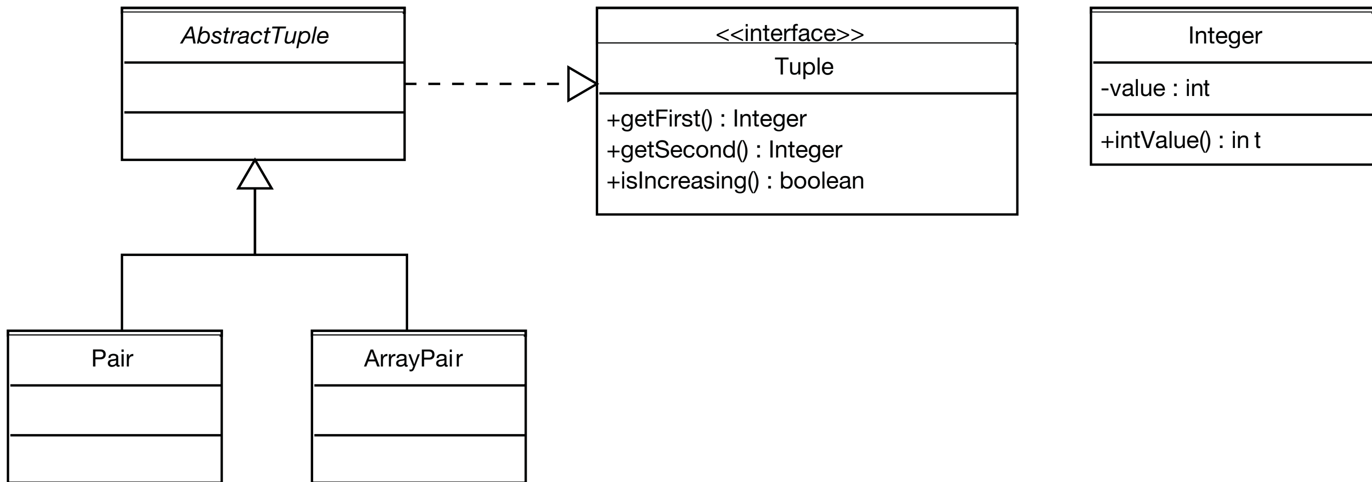
```java
int[] weights = { 10, 50, 30, 10, 40, 20, 10, 10 };
int[] map     = {  2,  0,  1,  0,  0,  1,  2,  2 };
System.out.println( Q1.getWeightDifference( weights, map, 3 ) );
```

Aucune validation des paramètres n'est requise. En particulier, **weights** et **map** désignent des tableaux dont les valeurs sont appropriées pour ce problème.

(page blanche)

Question 2 : (32 points)

Un tuple permet de sauvegarder deux nombres entier (des objets de la classe **Integer**). Tous les tuples ont une méthode **getFirst** ainsi qu'une méthode **getSecond** retournant une référence vers le premier et le second entier du tuple, respectivement. Un tuple possède une méthode **isIncreasing** qui retourne la valeur **true** si la valeur du premier élément est inférieure à celle du second élément, et **false** sinon.



Pour cette question, il y a une interface nommée **Tuple**, une classe abstraite nommée **AbstractTuple**, ainsi que deux implémentations concrètes, nommées **Pair** et **ArrayPair**. Vous trouverez leur description complète dans les pages qui suivent. La classe **Integer** possède une méthode nommée **intValue** qui retourne la valeur de l'entier sauvegardé dans cet objet. L'exécution des énoncés ci-dessous produira la sortie suivante : `7 is less than 17`.

```
Integer n1, n2, n3;

n1 = new Integer( 7 );
n2 = new Integer( 17 );
n3 = new Integer( 2 );

Tuple t1, t2;

t1 = new Pair( n1, n2 );
t2 = new ArrayPair( n1, n3 );

if ( t1.isIncreasing() ) {
    System.out.println( t1.getFirst() + " is less than " + t1.getSecond() );
}

if ( t2.isIncreasing() ) {
    System.out.println( t2.getFirst() + " is less than " + t2.getSecond() );
}
```

Assurez-vous d'implémenter tous les constructeurs et toutes les méthodes d'accès nécessaires à l'exécution des énoncés ci-dessus.

A. Donner l'implémentation de l'interface **Tuple**. Cette interface déclare 3 méthodes. Il y a deux méthodes d'accès, nommées **getFirst** et **getSecond**, qui toutes deux retournent une référence de type **Integer**. Finalement, l'interface déclare aussi une méthode nommée **isIncreasing** qui retourne un booléen. (6 points)

B. Concevoir la classe abstraite **AbstractTuple**. Elle réalise l'interface **Tuple**. Cette classe fournit une implémentation concrète de la méthode **isIncreasing**, qui retourne la valeur **true** si le premier nombre est un inférieur au second, et **false** sinon. (8 points)

C. Créer une implémentation concrète de la classe **AbstractTuple** que vous nommerez **Pair**. Cette implémentation possède deux variables d'instance. Ces variables sont des références vers le premier et le second nombre du tuple. Ajoutez tous les constructeurs et toutes les méthodes d'accès que vous jugez nécessaires. (8 points)

D. Vous devez concevoir une implémentation concrète de la classe **AbstractTuple** que vous nommerez **ArrayPair**. L'implémentation utilise un tableau de taille 2 afin de sauvegarder les références vers le premier et le second nombre du tuple. Ajoutez tous les constructeurs et toutes les méthodes d'accès que vous jugez nécessaires. (10 points)

Question 3 : (23 points)

A. En respectant les consignes présentées en classe, et décrites dans les notes de cours, dessinez les diagrammes de mémoire pour tous les objets et les variables locales de la méthode **Q3.test** à la suite de l'exécution de l'énoncé « **line = new Line(origin, new Point(11, 21)**) ».

```
public class Point {
    private int x = 0;
    private int y = 0;
    public Point( int x, int y ) {
        this.x = x;
        y = y;
    }
}

public class Line {
    private Point a;
    private Point b;
    public Line( Point a, Point b ) {
        this.a = a;
        this.b = b;
    }
}

public class Q3 {
    public static void test() {
        int zero;
        Point origin;
        Line line;
        zero = 0;
        origin = new Point( zero, 0 );
        line = new Line( origin, new Point( 11, 21 ) );
        // Ici!
    }
}
```

Réponse :

B. La classe **DynamicArrayStack** réalise l'interface **Stack** et utilise la technique du tableau dynamique présentée en classe. Ce qui permet à la structure de données (ici une pile) d'augmenter sa taille physique selon les besoins de l'application.

Pour l'implémentation partielle de la classe **DynamicArrayStack** ci-dessous, implémentez la méthode **trimToSize()** qui réduit la taille physique de la pile. La taille physique sera alors le maximum de minCapacity et la taille logique.

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

    // La taille minimum de cette pile

    private final int minCapacity;

    // La valeur de l'increment utilise lorsque la taille augmente

    private static final int DEFAULT_INCREMENT = 25;

    // Chaque fois que la pile est pleine, un nouveau tableau de
    // plus grande capacite est cree. La taille du nouveau
    // tableau est la taille de l'ancien plus la valeur de
    // l'increment.

    private final int increment;

    // Une reference vers le tableau utilise pour sauvegarder les
    // elements de la pile

    private E[] elems;

    // Une variable d'instance pour memoriser la position du
    // dessus de la pile dans le tableau. Lorsque la pile est vide,
    // sa valeur est -1.

    private int top = -1;

    // Implementez la methode void trimToSize() qui reduit la
    // taille physique au minimum, soit le maximum entre la
    // capacite minimale et la taille logique.

    public void trimToSize() {

    }
}
```

C. Donnez le résultat qui sera affiché sur la sortie standard à la suite de l'exécution de la méthode **main**.

```
public static void main( String[] args ) {

    Stack<Integer> s, t;

    s = new DynamicArrayStack<Integer>( 100 );

    for (int i=1; i<5; i++) {
        s.push( new Integer( i ) );
    }

    Integer x = null;

    if ( ! s.isEmpty() ) {
        x = s.pop();
    }

    t = new DynamicArrayStack<Integer>( 100 );

    while ( ! s.isEmpty() ) {
        t.push( s.pop() );
    }

    t.push( x );

    while ( ! t.isEmpty() ) {
        System.out.print( t.pop() );
        if ( ! t.isEmpty() ) {
        System.out.print( "," );
        }
    }
    System.out.println();

}
```

Réponse :

D. Si le constructeur d'une sous-classe ne fait pas un appel explicite au constructeur de la super-classe.

(a) une erreur d'exécution surviendra

(b) ceci causera une erreur de compilation

(c) le constructeur fera un appel implicite

(d) la classe est implicitement déclarée abstraite

(e) aucune de ces réponses

Réponse :

E. Toutes les classes en Java sont des sous-classes de :

(a) String
(b) java.lang
(c) Java
(d) Class
(e) Object

Réponse :

F. Étant donné l'énoncé qui suit :

```
Comparable s = new String();
```

Quels sont les énoncés qui sont vrai :

(a) Il y aura une erreur de compilation.
(b) Il y aura une erreur d'exécution.
(c) Un objet de la classe **String** sera désigné par une référence de type **Comparable**.
(d) Bien que l'énoncé soit valide, on devrait l'éviter puisqu'il porte à confusion.
(e) Aucun des énoncés n'est vrai.

Réponse :

G. Donnez le résultat affiché sur la sortie standard lorsque la méthode **main** sera exécutée.

```
public class Ticket {
    private int nextSerialNumber = 100;
    private int serialNumber;
    public Ticket() {
        serialNumber = nextSerialNumber;
        nextSerialNumber = nextSerialNumber + 1;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
    public static void main( String[] args ) {
        Ticket t1, t2, t3;
        t1 = new Ticket();
        t2 = new Ticket();
        t3 = new Ticket();
        System.out.print( t1.getSerialNumber() + "," );
        System.out.print( t2.getSerialNumber() + "," );
        System.out.println( t3.getSerialNumber() );
    }
}
```

Réponse :

H. Donnez le résultat affiché sur la sortie standard lorsque la méthode **main** sera exécutée.

```
public class Ticket {
    private static int nextSerialNumber = 100;
    private static int serialNumber;
    public Ticket() {
        serialNumber = nextSerialNumber;
        nextSerialNumber = nextSerialNumber + 1;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
    public static void main( String[] args ) {
        Ticket t1, t2, t3;
        t1 = new Ticket();
        t2 = new Ticket();
        t3 = new Ticket();
        System.out.print( t1.getSerialNumber() + "," );
        System.out.print( t2.getSerialNumber() + "," );
        System.out.println( t3.getSerialNumber() );
    }
}
```

Réponse :

I. Donnez le résultat affiché sur la sortie standard lorsque la méthode **main** sera exécutée.

```
public class Ticket {
    private static int nextSerialNumber = 100;
    private int serialNumber;
    public Ticket() {
        serialNumber = nextSerialNumber;
        nextSerialNumber = nextSerialNumber + 1;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
    public static void main( String[] args ) {
        Ticket t1, t2, t3;
        t1 = new Ticket();
        t2 = new Ticket();
        t3 = new Ticket();
        System.out.print( t1.getSerialNumber() + "," );
        System.out.print( t2.getSerialNumber() + "," );
        System.out.println( t3.getSerialNumber() );
    }
}
```

Réponse :

(page blanche)