

UQÀM Université du Québec à Montréal

Programmation II (inf-2120)

Été 2013

Examen intra

11 juin 2013

CONSIGNES

· **Les règlements de l'UQAM concernant le plagiat seront strictement appliqués.**

· Il est important de bien expliquer vos choix s'il y a lieu.

· Aucune documentation permise.

· La durée de l'examen est de 3 heures.

· Vous pouvez utiliser les versos comme brouillon ou comme espace supplémentaire.

· **Il est interdit de dégrafer le questionnaire.**

· Les téléphones cellulaires, calculatrices, ordinateurs, palm, baladeurs, iPods, etc. sont interdits.

#1 _____ / 14

#2 _____ / 16

#3 _____ / 14

#4 _____ / 14

#5 _____ / 14

#6 _____ / 14

#7 _____ / 14

IDENTIFICATION

NOM : _____

PRÉNOM : _____

CODE PERMANENT : _____

SIGNATURE : _____

GROUPE : _____

PROFESSEUR : _____

TOTAL

_____ / 100

commentaire :

Numéro 1. (14 pts)

Objectif(s) :

- Application des connaissances.
- Héritage et portée.
- Lecture de code.

Soit les déclarations suivantes (toutes dans le même **package**) :

```
public class A{
    public int j;
    protected int k;
    private int n;
}
public class B extends A{
    public int p;
    protected int r;
    private int t;
}
public class C extends B{
    public int v;
    protected int x;
    private int z;
}
```

a) (8 pts) Encerchez les noms des variables d'instances qui sont visibles dans la classe C.

j k n p r t v x z

b) (3 pts) Serait-il légal d'ajouter la déclaration suivante dans la classe C :

```
public int r;
```

Encerchez votre choix :

oui non

c) (3 pts) La méthode suivante est-elle permise :

```
public class D {
    public static A nouveau() {
        return new C();
    }
}
```

Encerchez votre choix :

oui non

Numéro 2. (16 pts)

Objectif(s) :

- Application des connaissances.
- Héritage, constructeur.
- Lecture et écriture de code.

Soit la classe suivante :

```
public class Tableau {
    /**
     * Construit un Tableau à taille fixe.
     * @param taille Nombre d'élément que va contenir le tableau. Le
     * comportement du constructeur n'est pas défini si la taille est négative.
     */
    public Tableau( int taille )
    /**
     * Lit un élément du tableau.
     * @param position Une valeur de 0 à taille - 1 indiquant la position de
     * l'élément à lire. Le comportement n'est pas défini si la position est
     * négative ou plus grande ou égal à la taille.
     * @return La valeur lue.
     */
    public int element( int position )
    /**
     * Place une valeur dans le tableau.
     * @param position Une valeur de 0 à taille - 1 indiquant la position de
     * l'élément à assigner. Le comportement n'est pas défini si la position
     * est négative ou plus grande ou égal à la taille.
     * @param element Valeur placé dans le tableau.
     */
    public void assigner( int position, int element )
    /**
     * Retourne le nombre de cases que contient le tableau.
     * @return La taille du tableau.
     */
    public int taille()
}

```

Nous voulons écrire une classe qui répondent aux critères suivants :

```
/**
 * Cette classe représente un Tableau à décalage. C'est un tableau de taille
 * fixe (choisi lors de la création) qui permet de déplacer les éléments vers
 * la gauche.
 */
public class TableauDecalage extends Tableau

```

a) (8 pts) Écrivez le code pour le constructeur suivant :

```
/**
 * Construit un tableau permettant les décalages vers la gauche.
 * @param taille Nombre d'élément que va contenir le tableau. Le
 * comportement du constructeur n'est pas défini si la taille est négative.
 */
public TableauDecalage( int taille )
```

b) (8 pts) Écrivez le code pour la méthode suivante :

```
/**
 * Déplace les éléments du tableau vers la gauche. Les éléments
 * qui se retrouveront hors du tableau sont perdus. Les cases vides à
 * la droite du tableau sont remplies avec la valeur en argument.
 * @param decalage Indique le nombre de case pour le déplacement à gauche.
 * @param valeur Valeur par défaut qui remplacera les valeurs à droite.
 * Exemple :
 * Le tableau t de taille 5 contenant : 3, 4, 1, 6, 5
 * nous appliquons un décalage de 2 cases vers la gauche et les cases
 * vide à droite seront remplies avec la valeur 8 :
 * t.decalageGauche( 2, 8 );
 * Après l'appel le tableau t contiendra : 1, 6, 5, 8, 8
 *
 * Les valeurs 3 et 4 sont éjecter du tableau (perdues), les valeurs
 * 1, 6, 5 sont décale de 2 cases et la valeur 8 est placée dans les
 * cases à droite.
 */
public void decalageGauche( int decalage, int valeur )
```

Numéro 3. (14 pts)**Objectif(s) :**

- Synthèse de la matière.
- Classe abstraite.

Question : Expliquez le fonctionnement des classes abstraites. Ce quelles impliquent et demandent aux classes qui en héritent.

Numéro 4. (14 pts)**Objectif(s) :**

- Application des connaissances.
- Héritage et méthode.

Écrivez ce que va afficher ce code lors de l'exécution :

```
public class PeutEtre {
    public Integer dePeutEtre( Integer d ) {
        return null;
    }
}
public class Seulement extends PeutEtre {
    private Integer contenu;
    public Seulement( Integer c ) {
        contenu = c;
    }
    @Override
    public Integer dePeutEtre( Integer d ) {
        return contenu;
    }
}
public class Rien extends PeutEtre {
    @Override
    public Integer dePeutEtre( Integer d ) {
        return d;
    }
}
public class P {
    public static void main(String[] args) {
        PeutEtre [] r = new PeutEtre[5];
        r[0] = new Rien();
        r[1] = new Seulement( 1 );
        r[2] = new Seulement( 2 );
        r[3] = new Rien();
        r[4] = new Seulement( 3 );
        for( int i = 0 ; i < 5 ; ++ i ) {
            System.out.println( r[i].dePeutEtre( 0 ) );
        }
    }
}
```

Numéro 5. (14 pts)**Objectif(s) :**

- **Analyse de problème.**

Soit les classes suivantes :

```
public class Objet2D           { ... }
public class Carre    extends Objet2D { ... }
public class Objet3D           { ... }
public class Plan     extends Objet3D { ... }
public class Sphere   extends Objet3D { ... }
public class Calcule           { ... }
```

Nous voulons placer une méthode statique 'aire' dans la classe `Calcule`. Cette méthode va recevoir un argument duquel elle va calculer l'aire. Nous voulons limiter les types de classes permissent comme argument à la méthode 'aire', seulement `Carre` et `Sphere` pourront être admises. Expliquez comment nous pouvons faire cela. Décrivez en détail votre technique. Vous ne pouvez pas modifier la hierarchie de classes déjà existante.

Numéro 6. (14 pts)**Objectif(s) :**

- Application des connaissances.
- Pile (Queue).

Écrivez ce que va afficher ce code lors de l'exécution :

```
import java.util.Stack;

public class P {
    public static void main(String[] args) {
        Stack<Integer> pile = new Stack<>();
        int i = 0;
        while( i < 7 ) {
            pile.push(i);
            ++ i;
            pile.push(i);
            ++ i ;
            pile.pop();
        }
        while( ! pile.empty() ) {
            System.out.println( pile.peek() );
            pile.pop();
        }
    }
}
```


Numéro 7. (14 pts)

Objectif(s) :

- Application des connaissances.
- Interface.
- Type générique.

Soit l'interface suivant :

```
public interface Comparable {
    public enum Ordre { PLUS_PETIT, EGAL, PLUS_GRAND }
    Ordre comparer( Comparable v2 );
}
```

Nous écrivons une classe Couple qui contient deux éléments de même type.

```
public class Couple< T extends Comparable > implements Comparable {
    private T a1;
    private T a2;
    public Couple( T v1, T v2 ) {
        a1 = v1;
        a2 = v2;
    }
}
```

Écrivez le code de la méthode `comparer` ajoutée à cette classe.

```
/**
 * Deux couples sont comparés de la façon suivante :
 * Si la composante a1 du premier est PLUS_PETIT que la composante a1
 * du deuxième alors le résultat sera PLUS_PETIT.
 * Si la composante a1 du premier est PLUS_GRAND que la composante a1
 * du deuxième alors le résultat sera PLUS_GRAND.
 * Si la composante a1 du premier est EGAL à la composante a1 du
 * deuxième alors la méthode retourne le résultat de la comparaison
 * des composantes a2 des deux couples.
 * @param v2 Deuxième couple de la comparaison. ('this' étant le premier)
 */
@Override
public Comparable.Ordre comparer(Couple< T > v2)
```