

POO-IG

Programmation Orientée Objet et Interfaces Graphiques

Examen de session 1

8 Janvier 2019

Durée : 2 heures

Documents autorisés : trois feuilles A4 recto-verso manuscrites ou imprimées. Portables/Ordinateurs/Tablettes interdits.

IMPORTANT : Dans les questions à choix multiple vous devez **entourer toutes les réponses correctes** (il peut y en avoir plusieurs). Si l'ensemble des réponses entourées ne coïncide pas avec l'ensemble des réponses correctes la question ne donnera aucun point.

Dans toutes les questions les étoiles donnent une indication approximative de la difficulté (plus d'étoiles = exercice plus difficile).

Question 1 (*)

```
class A {
    int c = 0;
    public int getC () {
        return c;
    }
}
class B extends A {
    int c = 1;
}
public class Test {
    public static void main (String args[]) {
        B b = new B();
        System.out.print(b.c+" ");
        System.out.println(b.getC());
    }
}
```

Qu'affiche le programme ci-dessus ?

Réponses possibles :

- a. 0 1
- b. 1 0
- c. 0 0
- d. 1 1

Question 2 (**)

Soit A et B deux classes définies comme suit :

```
public class B{}
public class A extends B{}
```

Entourer les réponses correctes :

Réponses possibles :

- a. ArrayList est un sous-type de ArrayList<? extends B>,
- b. ArrayList<A> est un sous-type de ArrayList<? super B>,
- c. ArrayList<A> est un sous-type de ArrayList,
- d. ArrayList est un sous-type de ArrayList<? super A>.

Question 3 (*)

Lorsqu'une classe C étend une classe B qui étend une classe A est-ce que : (entourer les réponses correctes)

Réponses possibles :

- a. toutes les instances de C sont aussi des instances de A
- b. toutes les instances de A sont aussi des instances de C
- c. toutes les instances de B sont aussi des instances de C
- d. les trois réponses précédentes sont fausses.

Question 4 (*)

```
class Portable {
    public void sonne(Sonnerie s) { s.sonne(); }
}
class Sonnerie {
    public void sonne() { System.out.println("Z Z"); }
}
class SimpleSonnerie extends Sonnerie{
    public void sonne() { System.out.println("Z"); }
}
public class Bruit {
    public static void main(String[] args) {
        Portable tel = new Portable();
        Sonnerie s1 = new Sonnerie();
        SimpleSonnerie s2 = new SimpleSonnerie();
        Sonnerie s3 = s2;
        tel.sonne(s1);
        tel.sonne(s3);
        tel.sonne(s2);
        tel.sonne((Sonnerie)s2);
    }
}
```

Qu'affiche le programme ci-dessus ?

Réponses possibles :

- a. Z Z [retour à la ligne] Z Z [retour à la ligne] Z [retour à la ligne] Z
- b. Z Z [retour à la ligne] Z Z [retour à la ligne] Z Z [retour à la ligne] Z Z
- c. Z Z [retour à la ligne] Z [retour à la ligne] Z [retour à la ligne] Z Z
- d. Z Z [retour à la ligne] Z [retour à la ligne] Z [retour à la ligne] Z

Question 5 (*)

On considère le code suivant

```
public static void main(String args[]) {  
    try {  
        int a, b; b = 0; a = 5 / b;  
    } catch(ArithmeticException e) {  
        System.out.print("A ");  
    }  
    System.out.print("B ");  
}
```

Qu'est-il affiché à l'exécution du programme ?

Réponses possibles :

- a. A
- b. B
- c. A B
- d. Exception in thread "main" java.lang.ArithmeticException : division by zero

Question 6 (**)

On définit les classes ci-dessous :

```
public class Voiture{  
    public int kmParcours;  
}  
  
public class Ferrari extends Voiture{  
}
```

Quelles propositions parmi les suivantes sont vraies ?

Réponses possibles :

- a. On peut ajouter un constructeur dans les deux classes.
- b. Pour pouvoir créer des voitures et des Ferraris neuves (c'est-à-dire qui n'ont parcouru aucun kilomètre) il faut ajouter un constructeur dans les deux classes.
- c. On peut ajouter un constructeur dans Voiture qui construit des voitures avec un kilométrage non nul, sans ajouter de constructeur dans Ferrari.
- d. On peut ajouter un constructeur dans Ferrari qui construit des Ferraris avec un kilométrage non nul, sans ajouter de constructeur dans Voiture.

Question 7 ()**

Nous souhaitons représenter une version simplifiée de la carte vitale où nous supposerons que son numéro n'est composé que de la concaténation du sexe (2 pour un homme, 3 pour une femme), de l'année de naissance de l'individu à quatre chiffres (1998 par exemple) et du mois de naissance de l'individu à deux chiffres. Par exemple, la coupe du monde de juillet 1998 aurait le numéro 3199807. Nous disposons des classes suivantes :

```
public class Carte{
    int numero;

    public Carte(int numero){
        this.numero=numero;
    }
}

public class Individu{
    boolean femme;
    int anneeNaissance;
    int moisNaissance;

    public Individu(boolean femme, int an, int mois){
        this.femme = femme;
        this.anneeNaissance = an;
        this.moisNaissance = mois;
    }
}
```

Java dispose d'une interface fonctionnelle `Function<T,R>` qui n'a qu'une seule méthode `R apply(T t)`.

Déclarer une variable `nouvelIndividu` du bon type et l'initialiser à l'aide d'une expression lambda. Cette expression lambda doit décrire une fonction qui associe à une carte vitale (au numéro respectant la norme décrite ci-dessus) l'individu ayant cette carte. (Vous pourrez utiliser les méthodes `Integer.parseInt(s)` pour convertir une chaîne de caractères `s` en entier, et `Integer.toString(n)` pour convertir un entier `n` en chaîne de caractères. Vous pourrez également utiliser `s.substring(i, j)` pour lire la sous-chaîne de `s` entre les indices `i` (inclus) et `j` (exclu).)

Question 8 (**)

On considère l'extrait de code suivant :

```
public static float prodFloat(List<? super Number> list) {
    float x = 1;
    for (Number y : list)
        x *= y.floatValue();
    return x;
}
```

Cet extrait de code compile-t-il ? Expliquez pourquoi.

Question 9 (*)

Soit A et B deux classes définies comme suit :

```
public class B{}
public class A extends B{}
```

Entourer les réponses correctes.

Réponses possibles :

- a. toutes les méthodes de la classe A sont héritées par la classe B,
- b. toutes les méthodes de la classe B sont héritées par la classe A,
- c. toutes les méthodes de la classe B sont accessibles par la classe A,
- d. toutes les méthodes de la classe A sont accessibles par la classe B.

Question 10 (*)

Écrire une méthode statique qui prend en argument un tableau t et qui renvoie le dernier élément de t . On ne suppose rien sur le type des éléments du tableau.

Question 11 ()**

Quelles affirmations suivantes sont correctes ?

Réponses possibles :

- a. Une classe abstraite peut avoir une méthode “final”
- b. Une classe implémentant une interface doit implémenter/redéfinir toutes les méthodes déclarées dans l'interface
- c. Dans une interface, toutes les paires possibles des modifieurs “private”, “abstract” et “final” sont incompatibles dans une même déclaration de méthode
- d. Une classe abstraite peut contenir une méthode “private”.

Question 12 (*)**

Écrire une expression lambda à la place du commentaire pour qu'à l'exécution de la méthode `main`, le caractère `c` vaille '3'.

```
public interface Conv<A,B>{
    B conv(A a);
}
public class Barbara<A,B,C>{
    Conv<Conv<A,B>, Conv< A, Conv<Conv<B,C>, C>>> barbara =
    // Votre expression lambda ici
}
public class Test {
    public static void main(String[] args) {
        Conv<Integer,String> f = i -> Integer.toString(i);
        Conv<String,Character> g = s -> s.charAt(0);
        Barbara<Integer,String,Character> barb =
        new Barbara<Integer,String,Character>();
        int test = 3345;
        char c = barb.barbara.conv(f).conv(test).conv(g);
        // c doit valoir '3'
    }
}
```

Votre expression lambda :

Question 13 (*)

Soit `B` la classe définie comme suit :

```
public class B {  
    public class A {}  
}
```

Quelle est la bonne façon d'instancier A depuis l'extérieur de B ?

Réponses possibles :

- a. `B.A b = new B.A()`
- b. `A.B a = b.new A()`, avec b une instance de B
- c. `A a = b.new A()`, avec b une instance de B
- d. `B.A a = b.new A()`, avec b une instance de B

Question 14 (**)

L'interface fonctionnelle `ToIntBiFunction<T,U>` (du package `java.util.function`) contient pour unique méthode `int applyAsInt(T value1, U value2)`. On souhaite utiliser cette interface pour calculer la taille au carré d'une chaîne de caractères. Pour cela, on considère le code ci-dessous :

```
XXXX squareLength = YYYY;  
String s = "hello";  
System.out.println(squareLength.applyAsInt(s,s)); // affiche 25
```

Que faut-il indiquer à la place de XXXX et YYYY pour que le fragment de code ci-dessus affiche la taille au carré de s (25 en l'occurrence) ? Indice : YYYY doit contenir une expression lambda.

POO-IG

Programmation Orientée Objet et Interfaces Graphiques

Examen de session 1

9 Janvier 2018

Durée : 2h

*Documents autorisés : trois feuilles A4 recto-verso manuscrites ou imprimées. Portables interdits.
Le barème est donné à titre indicatif.*

IMPORTANT : Dans les questions à choix multiples, plusieurs options peuvent être des réponses valides. Cependant vous devez **entourer au plus une option**. Si plusieurs options sont entourées, la question sera considérée sans réponse.

Question 1 (1 point)

Soit le fragment de code suivant :

```
class Ville {
    private String nom; private double latitude; private double longitude;
    public Ville (String nom, double lat, double lon)
    { this.nom = nom; this.latitude = lat; this.longitude = lon;}
    public double distance (Ville v) {...} //renvoie le nombre de Km entre this et v
    public String getNom () {return nom;}
    public void setNom (String nom) {this.nom = nom;}
    public double getLatitude() {return latitude;}
    public void setLatitude (double l) {latitude = l;}
    public double getLongitude() {return longitude;}
    public void setLongitude (double l) {longitude = l;}
}
class SegmentCourt {
    private Ville depart; private Ville arrivee;
    public SegmentCourt (Ville d, Ville a) {
        if (d.distance(a) > 1000) throw new RuntimeException();
        this.depart = d; this.arrivee = a;
    }
}
```

Et l'utilisation suivante de la classe `SegmentCourt`

```
Ville v1 = new Ville("Paris", 48.85, 2.35);
Ville v2 = new Ville("Berlin", 52.52, 13.40);
SegmentCourt s = new SegmentCourt (v1, v2);
```

Laquelle des options suivantes viole le principe d'encapsulation de la classe `SegmentCourt` ?

Réponses possibles :

- a. `String n = v1.getNom();`
- b. `v1.setNom("Lutece");`
- c. `v2.setLatitude (52.53);`
- d. aucune des options.

Question 2 (1 point)

Etant donnée la définition de classe suivante

```
public class Q {
    int f (int n){
        int a = x;
        for (int x = 1; x < n; x++) { a = a * x; }
        a = a * x;
        return a;
    }
    private int x;
    Q (int x) { this.x = x;}
}
```

Etant données deux variables entières m et n initialisées, et les instructions suivantes :

```
Q q = new Q(m); System.out.println (q.f(n));
```

Quelle est la valeur affichée ?

Réponses possibles :

- a. Rien. Erreur à la compilation de la classe Q
- b. $n! m$
- c. m^{n+1}
- d. $(n - 1)! m^2$

Question 3 (1 point)

Etant donnée la classe ville de la Question 1, laquelle des définitions suivantes est correcte ?

Réponses possibles :

a.

```
class VilleDansRegion extends Ville {
    static final String region;
    VilleDansRegion ( String nom, double lat, double lon, String r )
    { super(nom, lat, lon); region = r; }
}
```

b.

```
class VilleDansRegion extends Ville {
    static final String region;
    {region = "Ile de France";}
    VilleDansRegion ( String nom, double lat, double lon )
    { super(nom, lat, lon); }
}
```

c.

```
class VilleDansRegion extends Ville {
    static final String region = "Ile de France";
    VilleDansRegion ( String nom, double lat, double lon, String r )
    { region = r; super(nom, lat, lon); }
}
```

d. Aucune

Question 4 (3 points)

Etant donné le fragment de code ci-dessous :

```
interface Taggable<T> {
    void tag(T t);
    T getTag();
}

class Elem implements Taggable<Elem> {...}
class Obj implements Taggable<Obj> {...}
class RedElem extends Elem {...}

class Q {
    static <XXXX> void idTag (List<T> l) {
        for (T e : l) e.tag (e);
    }
    public static void main (String args[]) {
        ArrayList<Elem> a = new ArrayList<Elem> ();
        ArrayList<RedElem> b = new ArrayList<RedElem> ();
        ArrayList<Obj> c = new ArrayList<Obj> ();
        ...// initialisation de a, b et c
        idTag (a); idTag (b); idTag (c);
    }
}
```

Donner la déclaration du type T qui doit remplacer XXXX dans la méthode idTag (donner la réponse dans l'espace ci-dessous)

XXXX :

Question 5 (3 points)

Etant donné le fragment de code ci-dessous :

```
interface Membre <S, R> {
    R action (S s);
}

public class Q {
    public static <XXXX> List<T> apply (List<T> l, S s) {
        List<T> lr = new ArrayList<T> ();
        for (T elem : l) lr.add (elem.action (s));
        return lr;
    }
}
```

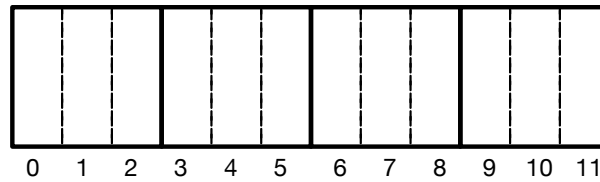
Donner la déclaration de type qui doit remplacer XXXX dans la méthode apply (donner la réponse dans l'espace ci-dessous)

XXXX :

Question 6 (12 points)

Le but de cette question est d'écrire **un package tape** qui fournit des classes et interfaces pour la simulation (simplifiée) d'une mémoire à accès séquentiel.

FIGURE 1 – Une mémoire comportant 4 blocs de 3 bytes chacun



Le package doit contenir au moins :

1. **Une interface générique** `Moveable` qui intuitivement représente des objets qui peuvent être déplacés le long d'une dimension.
2. **Une exception** `EOTException` (pour *End Of Tape Exception*) de type "checked", qui signale le tentatif de lire en dehors des limites de la mémoire à accès séquentiel.
3. **Une classe** `Tape` qui représente une mémoire à accès séquentiel, avec **une classe interne** `TapePointer` qui décrit des têtes de lecture/écriture.
4. **Une classe** `Q` d'utilité et test.

L'interface `Moveable` doit contenir :

- une méthode `move` censée déplacer l'objet sur laquelle elle est invoquée d'un nombre entier d'unités ;
- une méthode `retrieve` censée retourner un tableau du type variable (intuitivement cela représente des données récupérées à la position courante de l'objet).

La classe `Tape` représente une mémoire à accès séquentiel comme un tableau de `Byte` (cf. Figure 1). La mémoire est divisée en blocs consécutifs, tous de la même taille (c'est-à-dire tous comportant le même nombre de cases du tableau).

Une mémoire séquentielle de b blocs se remplit bloc par bloc, de gauche à droite, à partir du bloc 0, jusqu'au bloc b .

En plus de représenter la mémoire et sa subdivision en blocs, la classe `Tape` doit contenir au moins :

- l'indice du dernier bloc utilisé (-1 si aucun bloc n'a été écrit) ;
- une méthode `append` qui écrit un nouveau bloc à la fin de la partie utilisée de la mémoire ;
- une classe interne `TapePointer` qui représente des pointeurs capables de parcourir les blocs de la mémoire ;
- une tête de lecture/écriture `head`, de type `TapePointer`, qui pointe au dernier bloc utilisé (`null` si aucun bloc n'a été écrit) ;
- Une méthode `tapeForwardReader()` qui renvoie un `TapePointer` initialisé sur le bloc 0 (ou `null` si la mémoire est vide).
- Une méthode `tapeReverseReader()` qui renvoie un `TapePointer` initialisé sur le dernier bloc de la mémoire (ou `null` si la mémoire est vide).

La classe interne `TapePointer` implémente l'interface `Moveable` du type approprié, et maintient un pointeur à un bloc (c-à-d l'indice du premier byte du bloc pointé).

En plus des méthodes de l'interface `Moveable`, la classe interne doit fournir au moins une méthode `write`.

- La méthode `move` déplace le pointeur du nombre de blocs passé en paramètre.
- La méthode `retrieve` renvoie le bloc sous le pointeur.
- La méthode `write` écrit sur le bloc pointé un nouveau contenu (de taille correspondante) passé en paramètre.

Contraintes à respecter :

- Au contraire de l'interface `Moveable`, La classe `TapePointer` doit être visible uniquement dans la classe `Tape`.
- Un `TapePointer` peut uniquement pointer à un bloc utilisé (c'est à dire d'indice compris entre 0 et le dernier bloc utilisé). Veiller à vérifier cette contrainte à la fois à la création et à chaque déplacement du pointeur. Toute violation doit soulever une exception `EOTException`.
- La classe `Tape` écrit sur la mémoire uniquement grâce à la méthode `write` de la tête de lecture/écriture `head`.

La classe `Q` doit contenir au moins trois méthodes statiques, 1) `forwardScan`, 2) `backwardScan` et 3) `evenScan` qui affichent sur la console le contenu d'un `Tape` passé en paramètre, respectivement : 1) du début à la fin, 2) de la fin au début, et 3) uniquement les blocs pairs du début à la fin.

Pour parcourir la mémoire, ces méthodes récupèrent et se servent d'un `TapePointer` approprié. De plus elles utilisent l'exception `EOTException` pour détecter que les limites de la mémoire ont été atteints.

La classe `Q` contient enfin une méthode `main` qui initialise le contenu d'un `Tape` avec des appels successifs de `append`, puis teste les trois méthodes statiques ci-dessus.