POO-IG
Programmation Orientée Objet et Interfaces Graphiques
Examen de session 1

8 Janvier 2019

Durée : 2 heures
(Correction)

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

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

Qu'affiche le programme ci-dessus ?

Réponses possibles :

- → 0 1
- × 1 0
- × 1 1
- × 0 0

Question 2 (***)

Écrire une expression lambda à la place du commentaire pour qu'à l'execution 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<A, Conv<Conv<A,B>, 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(test).conv(f).conv(g);
        // c doit valoir '3'
  }
}
```

Votre expression lambda :

x -> f -> g -> g.conv(f.conv(x));

Question 3 (*)

Soit `A` la classe définie comme suit :

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

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

Réponses possibles :

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

Question 4 (**)

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 le produit de la taille d'une chaîne de caractères et d'un entier. Pour cela, on considère le code ci-dessous :

```
XXXX lengthProduct = YYYY;
String s = "hello"; int i = 5;
System.out.println(lengthProduct.applyAsInt(s,i)); // affiche 25
```

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

*ToIntBiFunction<String, Integer> lengthProduct = (s, i) -> s.length()*i;*

Question 5 (*)

On considère le code suivant

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

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

Réponses possibles :

× B A

× A B

× Exception in thread "main" java.lang.ArithmeticException : division by zero

→ B

L'exception est correctement gerée, d'où l'affichage de 'B'.

Question 6 (**)

On définit les classes ci-dessous :

```
public class Voiture{
  public int kmParcourus;
}

public class Ferrari extends Voiture{
}
```

Quelles propositions parmi les suivantes sont vraies ?

Réponses possibles :

× On peut ajouter un constructeur dans les deux classes, uniquement sous réserve que les deux aient le meme nombre d'arguments.

× On peut ajouter un constructeur dans Voiture qui construit des voitures avec un kilométrage non nul, sans ajouter de constructeur dans Ferrari.

→ On peut ajouter un constructeur dans Ferrari qui construit des Ferraris avec un kilométrage non nul, sans ajouter de constructeur dans Voiture.

× 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.

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 (H pour un homme, F pour une femme), de l'année de naissance de l'individu (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 $F199807$. Nous disposons des classes suivantes :

```
public class Carte{
  String numero;

  public Carte(String 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 `nouvelleCarte` du bon type et l'initialiser à l'aide d'une expression lambda. Cette expression lambda doit décrire une fonction qui associe à un individu une nouvelle carte vitale (au numéro respectant la norme décrite ci-dessus). *(Vous pourrez utiliser les méthodes `Integer.toString(i)` pour convertir un entier `i` en chaîne de caractères.)*

```
Function<Individu,Carte> nouvelleCarte= (bob -> {
    String aux="";
    if(bob.femme){
        aux+="F";
    }else{
        aux+="H";
    }
    aux+=Integer.toString(bob.anneeNaissance);
    if (bob.moisNaissance < 10) aux += "0";
    aux+=Integer.toString(bob.moisNaissance);
```

```
            return new Carte(aux);
        });
```

Question 8 (*)

Soit `A` et `B` deux classes définies comme suit :

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

Entourer les réponses correctes.

Réponses possibles :

- → toutes les méthodes de la classe `A` sont héritées par la classe `B`,
- × toutes les méthodes de la classe `B` sont accessibles par la classe `A`,
- × toutes les méthodes de la classe `A` sont accessibles par la classe `B`,
- → la classe B peut déclarer un champ avec le même nom qu'un champ de la classe A.

Question 9 (*)

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

Qu'affiche le programme ci-dessus ?

Réponses possibles :

- × Z [retour à la ligne] Z [retour à la ligne] Z Z [retour à la ligne] Z Z
- → Z [retour à la ligne] Z Z [retour à la ligne] Z Z [retour à la ligne] Z Z
- × Z Z [retour à la ligne] Z [retour à la ligne] Z Z [retour à la ligne] Z
- × Z [retour à la ligne] Z Z [retour à la ligne] Z Z [retour à la ligne] Z

Question 10 (*)

Écrire une méthode statique qui prend en argument un tableau a et un entier i et qui renvoie l'élément de a en position `i+1`, ou `null` si la position `i+1` dépasse les limites du tableau. On ne suppose rien sur le type des éléments du tableau.

```
public static <T> T renvoie (T[] t, int i) {
    if (i >= t.length) return null;
    return t[i+1];
}
```

Question 11 (**)

Soit `A` et `B` deux classes définies comme suit :

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

Entourer les réponses correctes :

Réponses possibles :

- → `ArrayList<A>` est un sous-type de `ArrayList<? super A>`,
- → `ArrayList<B>` est un sous-type de `ArrayList<? extends A>`,
- × `ArrayList<B>` est un sous-type de `ArrayList<A>`,
- → `ArrayList<B>` est un sous-type de `ArrayList<? extends B>`.

Question 12 (**)

On considère l'extrait de code suivant :

```
public static float prodFloat(List<? extends Number> list) {
    float x = 1;
    for (Number y : list)
```

```
        x *= y.floatValue();
    return x;
}
```

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

Oui, cet extrait de code compile. La méthode `prodFloat` *accepte comme argument une* `List<T>`*, où* `T` *est un sous-type de* `Number`*. Les éléments de la liste peuvent donc implicitement être upcastés en* `Number`

Question 13 (**)

Quelles affirmations suivantes sont correctes ?

Réponses possibles :

→ Dans une interface, toutes les paires possibles des modifieurs ”private”, ”abstract” et ”final” sont incompatibles dans une même déclaration de méthode

× Une classe implémentant une interface doit implémenter/redéfinir toutes les méthodes déclarées dans l'interface

× Une interface peut avoir une méthode final

→ Une interface peut contenir une méthode private.

pour la seconde pensez à default, ou a une classe abstraite implémentant l'interface Remarque : dans une classe, même abstraite, ”private” et ”final” ensemble sont ok (même si inutiles)

Question 14 (*)

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

Réponses possibles :

→ toutes les instances de A sont aussi des instances de B

→ toutes les instances de A sont aussi des instances de C

→ toutes les instances de B sont aussi des instances de C

→ les trois réponses précédentes sont vraies.