(Bon courage !)

Université de Paris

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

3 Janvier 2022

Durée : 2 heures
(Correction)

Nom :

Prénom :

Numéro d'étudiant :

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). Pour chaque question à choix multiple

- entourer une bonne réponse donne 1/4 de point, ainsi que ne pas entourer une mauvaise réponse ;
- entourer une mauvaise réponse donne -1/4 de point, ainsi que ne pas entourer une bonne réponse ;

Une question qui, ainsi faisant, donnerait un nombre négatif de points sera notée 0.
De plus chaque question aura un coefficient. Dans toutes les questions les étoiles donnent une indication approximative de la difficulté (plus d'étoiles = exercice plus difficile = coefficient plus élevé).

Question 1 (*)

Quelles affirmations sont vraies ?

Réponses possibles :

× Une interface peut hériter d'une classe
× Une classe peut implémenter au plus une interface
→ Une classe peut hériter d'au plus une classe
→ Une classe peut implémenter plusieurs interfaces

Question 2 (***)

Soit l'interface suivante

```
interface Func <S,T> {
  T f(S s);
}
```

Lesquelles des expressions lambda suivantes peuvent être affectées à la variable `f` déclarée comme ci-dessous ?

`Func<Integer, Func<String, Integer>> f ;`

Réponses possibles :

- × `(Integer i, String s) -> s.length() * i`
- → `(i) -> (s) -> s.length()`
- → `(Integer i) -> (String s) -> s.length() * i`
- × `(i, s) -> s.length()`

Question 3 (*)

Quelles affirmations sont vraies ?

Réponses possibles :

- × Une méthode abstraite peut avoir un corps (contenant un comportement "par défaut" pour cette méthode).
- → Une classe abstraite peut hériter d'une classe non-abstraite.
- × Une classe abstraite ne peut pas hériter d'une classe abstraite.
- → Une classe abstraite peut implémenter une interface sans pour autant proposer d'implémentation pour toutes ses méthodes abstraites.

Question 4 (**)

On dispose d'une classe `Etudiant`, décrivant les étudiants d'une université, et de l'interface ci-dessous. Notre objectif est de déterminer, au sein d'un tableau `Etudiant[] univ`, quel est la `noteMoyenne` la plus élevée parmi les étudiants de moins de 25 ans.

```
public class Etudiant {

  private String nom;
  private int noteMoyenne, age;

  public Etudiant(String nom, int noteMoyenne, int age) {...}

}

public interface AvecValeur {

  public double valeur();

  public static double valeurMaximale(AvecValeur [] tableau) {

    double maximum = -1;
    for (AvecValeur c : tableau) {
      if (c.valeur() > maximum) maximum = c.valeur();
    }
    return maximum;
  }
}
```

Expliquer comment modifier la classe `Etudiant` pour atteindre cet objectif en invoquant la méthode `AvecValeur.valeurMaximale(univ)`.

Il faut que la classe implémente l'interface. Pour cela il est nécessaire d'ajouter `implements AvecValeur` *et d'implémenter la méthode* `valeur`*, qui renverra la noteMoyenne de l'employé si* `age <= 25` *et* `-1` *sinon. Enfin on écrira* `AvecValeur.valeurMaximale(univ)`*.*

Question 5 (***)

```
public class App
{
    static class A {
        void f(A a) {
            System.out.print("A.f(A) ");
        }
    }
    static class B extends A {
        @Override
        void f(A a) {
            System.out.print("B.f(A) ");
        }
        void f(B b) {
            System.out.print("B.f(B) ");
        }
    }
```

```
    public static void main( String[] args )
    {
        A a = new B();
        B b = (B)a;
        a.f(a);
        a.f(b);
    }
}
```

Qu'affiche le programme ci-dessus ?

Réponses possibles :

- × A.f(A) A.f(B)
- × A.f(B) A.f(B)
- × B.f(A) B.f(B)
- → B.f(A) B.f(A)

Question 6 (**)

```
public class App
{
    public static void f(int i) {
        if (i < 0) throw new Exception();
        else throw new RuntimeException();
    }
    public static void main( String[] args ) throws Exception
    {
        try {
            f(42);
        }
        catch (RuntimeException re) {
            System.out.print("RuntimeException ");
        }
        finally {
            System.out.print("Finally");
        }
    }
}
```

Que fait le programme ci-dessus ?

Réponses possibles :

- → Il échoue à la compilation
- × La compilation réussit, mais le programme affiche une erreur à l'exécution à cause d'une exception non rattrapée
- × La compilation réussit, et le programme termine correctement en affichant `RuntimeException Finally`
- × La compilation réussit, et le programme termine correctement en affichant `Finally`

Question 7 (**)

On suppose d'avoir une classe qui décrit une collection d'éléments de type `S` définie comme suit

```
public class MaCollection<S> {
  private S[] t;
  private class Streamer...
  ...
}
```

La classe `Streamer` est membre de la classe `MaCollection<S>`, et étend l'interface `Iterator<T>`. `T` représente intuitivement le type d'une nouvelle valeur calculée à partir des éléments de la collection, et peut être différent du type `S` de ces elements.

Quelle est la bonne définition de la classe membre `Streamer` ?

Réponses possibles :

- × `private class Streamer<S> extends Iterator<T> {...}`
- → `private class Streamer<T> extends Iterator<T> {...}`
- × `private class Streamer extends Iterator<T> {...}`
- × `private class Streamer <S,T> extends Iterator<T> {...}`

Question 8 (*)

On rappelle qu'en Java la mémoire pour stocker les objets est allouée dans le tas, alors que les références *vers* les objets sont stockées dans la pile d'exécution.
Considérez les déclarations suivantes,

```
int x;
String s;
```

La variable `x` a comme valeur possible un entier entre -2^{31} et $2^{31} - 1$. En sachant que la variable `s` est une référence, quelles sont toutes les valeurs possibles pour la variable `s` ?

Soit `NULL` *soit un addresse de memoire.*

Question 9 (***)

Considerez le code suivant.

```
class Test {
    public static void main(String args[]) {
       System.out.print((new Integer(2) == new Integer(2)) + ";");
        System.out.print((new Integer(2) == (Integer) 2) + ";");
        System.out.print((2.0 == (Double) ((double) 2)) + ";");
        System.out.print(((Double) ((double) ((int) 2.0)) == (Double) 2.0) + ";");
    }
}
```

Quand on execute la méthode `main`, quelle chaîne est affichée dans la sortie standard ?

Réponses possibles :

- × true ;false ;true ;false ;
- × true ;false ;false ;false ;
- → false ;false ;true ;false ;
- × true ;false ;true ;true ;

Question 10 (*)

On suppose d'avoir une classe qui décrit une collection d'éléments de type `T` définie comme suit

```
public class MaCollection<T extends Number> {
  private T[] t;
  ...
}
```

Est-ce que le constructeur suivant est possible pour cette classe ?
`public MaCollection () {t = new T[10];}`

Réponses possibles :

- × Oui
- → Non
- × Oui seulement si `T` hérite de `Number`
- × Oui seulement si `T` est une classe ayant un constructeur sans paramètres

Question 11 (*)

Le code suivant produit 3 affichages, pouvez-vous dire lesquels **ET** les expliquer

```
static class A {
  public boolean test(A a) {
    return true;
  }
}

static class B extends A {
  public boolean test(B b) {
    return false;
  }
}
...
public static void main(String[] args) {
  A a = new B();
  System.out.println(!a.test(new B()));

  Integer i = 2;
  Integer j = 2;
  System.out.println(i == j);

  String s = "HELLO";
  s.toLowerCase();
  System.out.println(s);
```

```
}
```

Répondez ici à la question

- `false` : *le typage des arguments est déterminé à la compilation*
- `false` : *ce sont des objets ...*
- `HELLO` : *les String sont finaux, c'est pourquoi* `toLower` *retourne une nouvelle chaine. En l'occurrence ici le résultat n'est pas traité, il est donc perdu.*

Question 12 (*)

Le code suivant ne compile pas, pouvez-vous expliquer pourquoi en quelques lignes, et proposer **deux** corrections différentes.

```
public class C {
  void f (int i) { }

  class D extends C {
    void f(int i) { }
  }

  public static void main(String[] args) {
    C a = new D();
    a.f(0);
  }
}
```

Répondez ici à la question

La classe D est une classe membre, elle a donc besoin d'une instance pour pouvoir exister. Ce qui échoue c'est la construction. Deux corrections sont possibles :

- *définir D comme étant statique*
- *écrire qq chose comme* `C a = new C().new D();`

NE PAS RETOURNER AVANT D'EN ETRE AUTORISÉ