(Bon courage !)

Université Paris Cité

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

4 Janvier 2023

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).

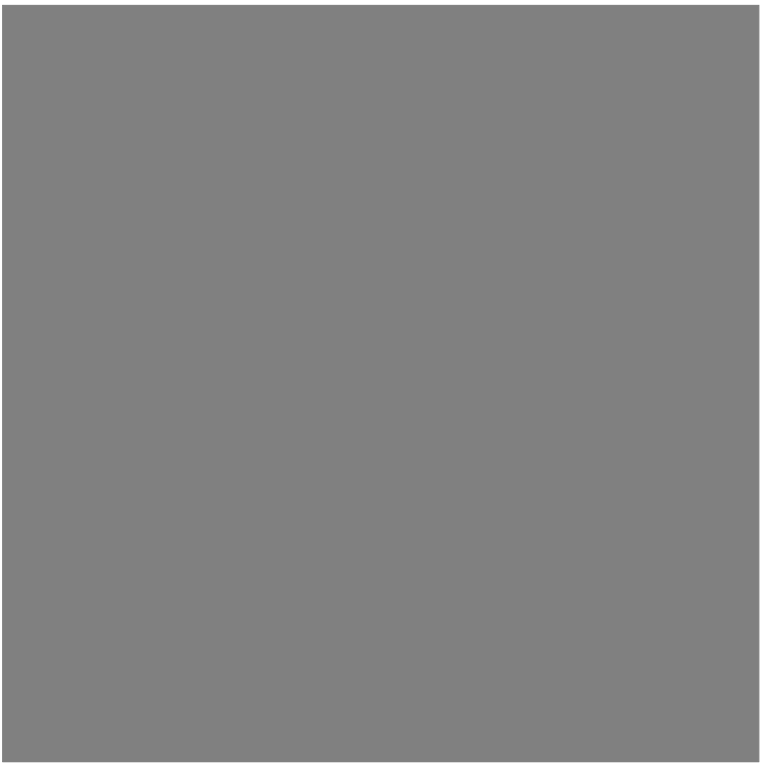
Question 1 (*)

```
class A {
  static class B {
    private int i = 1;
  }
  B b1 = new B();
  B b2 = new B();
  ...
}
```

Donner les réponses correctes

Réponses possibles :

- × `b1.i` et `b2.i` font référence au même champ (toute modification de l'un entraîne une modification de l'autre)
- → `b1.i` et `b2.i` font references à deux champs différents, qui peuvent être modifiés indépendamment
- → `b1.i` est visible dans la classe `A`
- × `b1` est un champ statique de la classe `A`



Question 2 (*)

Soit le programme suivant.

```
public class Example{
  static class A{
    private B b;
  }
  static class B extends A{
  }
  static class C extends B{
    public B func(){
      return new B();
    }
  }
  public static void main(String[] args){
    C c=new C();
    # d=c.func();
  }
}
```

Qu'est-ce qu'on peut avoir au lieu de "#" pour que le programme compile ?

Réponses possibles :

→ A.

→ B.

× C.

× Rien, il y a d'autres problèmes dans le code.

Question 3 (**)

Écrire une classe `Benchmark` avec une méthode `test` qui fait tourner 1000 fois un algorithme à tester passé en paramètre. Écrire également les autres classes / interfaces éventuellement nécessaires pour obtenir cela. Suivre le *strategy pattern*.

Écrire ici les classes/interfaces demandées

```
public class Benchmark {
  public void test (TestedAlgorithm a) {
    for (int i = 0; i < 1000; i++) a.run();
  }
}
public interface TestedAlgorithm {
  void run();
}

public class MonAlgo implements TestedAlgorithm{
  public void run() {...};
}
```

Question 4 (*)

Soient A,B,C les classes suivantes :

```
class A{
    public static void f(){
    System.out.println("Fonction:fA");
    }
```

```
    public static void h(){
    System.out.println("Fonction:hA");
    }
}
class B extends A{
    public static void f(){
      System.out.println("Fonction:fB");
    }
    public static void g(){
      System.out.println("Fonction:gB");
    }
}

class C extends B{
    public static void g(){
      System.out.println("Fonction:gC");
    }
    public static void h(){
      System.out.println("Fonction:hC");
    }
}

class test{
public static void main(String args[]){
  A s=new A();
  B t=new B();
  A u=new C();
  s.f();
  t.f();
  t.g();
  u.f();
  u.h();
}
}
```

Qu'affiche le programme précédent ?

Réponses possibles :

→
```
Fonction:fA
Fonction:fB
Fonction:gB
Fonction:fA
Fonction:hA
```

×
```
Fonction:fA
Fonction:fB
Fonction:gB
Fonction:fA
Fonction:hC
```

× le programme ne compile pas, l'instruction `u.h()` produit une erreur de compilation.

× le programme ne compile pas, l'instruction `u.f()` produit une erreur de compilation.

Question 5 (*)

```
B b = new B();
A a = b.m();
```

Si ce code ne pose pas de problème à la compilation, de quelle(s) information(s) peut-on être sûr ?

Réponses possibles :

× La classe `B` a un attribut nommé `m` qui est hérité de la classe `A`.

× La classe `B` hérite de la classe `A`.

→ `m` est une méthode dont le type de retour hérite de (ou est) la classe `A`.

× `m` est une méthode sans argument de la classe `A`.

Question 6 (**)

```
public class Test {

  public static class E1 extends Exception{}
  public static class E2 extends Exception{}

  public static void main(String[] args) throws E1,E2{
    try {throw new E1();
    }catch(Exception e){throw new E2();}
    finally {throw new E1();}
  }
}
```

Que va-t-il se passer à l'éxécution ?

Réponses possibles :

→ Seule une exception `E1` va être levée.

× Seule une exception `E2` va être levée.

× Une exception `E1` PUIS une exception `E2` vont être levées.

× Une exception `E2` PUIS une exception `E1` vont être levées.

Question 7 (**)

On considère le code suivant.

```
public class A implements I<A> {
  public int g(A a) { return 0; }
}

public class B extends A {}

public interface I<T> {
  public int g(T t);
}
```

Quelles sont les instructions qui compilent ?

Réponses possibles :

→ `I<A> i1 = new B();`

→ `I<A> i2 = (a -> a.g(a));`

× `I<B> i3 = new B();`

→ `I<B> i4 = (b -> b.g(b));`

Question 8 (**)

On considère la classe suivante.

```
public class Compteur <T> {
  public interface Predicate<T> {
    abstract boolean check(T t);
  }

  public int compte(T[] tab, Predicate<T> pred) {
    int s=0;
    for (T t : tab) if (pred.check(t)) s++;
    return s;
  }
}
```

En utilisant cette classe, écrire une fonction `int g(String [] tab)` qui renvoie le nombre d'éléments de `tab` qui commencent par le caractère "a".

```
public int g(String[] tab) {
  return new Compteur<String>().compte(tab,s -> s.length()>0 && s.charAt(0)=='a');
}
```

Question 9 (**)

Soit le programme suivant.

```
public class App {
    static class A {
        public A() { System.out.print("A"); }
        public A(String s) { this(); System.out.print(s); }
    }
    public static void main(String args[]) {
        A a = new A("B");
    }
}
```

Laquelle ou lesquelles des affirmations suivantes sont vraies ?

Réponses possibles :

- × Ce programme ne compile pas.
- × Ce programme compile, et affiche "A".
- × Ce programme compile, et affiche "B".
- → Ce programme compile, et affiche "AB".

Question 10 (**)

On considère l'interface suivante. Un objet `o` implémentant cette interface représente la fonction de `T` vers `R` associant à `t` la valeur `o.apply(t)`. On souhaite que la seconde méthode, `map`, ait par défaut le comportement suivant : qu'elle construise et renvoie la liste contenant, à chaque position `i`, l'image (selon la fonction representée) de élément `t` en position `i` de `list`.

```
interface Function <T, R> {
  R apply(T t);
  LinkedList<R> map(LinkedList<T> list);
}
```

Écrire cette implémentation par défaut pour `map`.

```
default LinkedList<R> map(LinkedList<T> list) {
  LinkedList<R> res = new LinkedList<>();
  for (T t : list) {
    res.add(apply(t));
  }
  return res;
}
```

Question 11 (***)

Le package `java.util.function` fournit les interfaces `Function` et `BiFunction` définies comme suit :

```
interface Function<T,R> {
    R apply(T t);
}
interface BiFunction<T,U,R> {
    R apply(T t, U r);
}
```

Mathématiquement, `Function<T,R>` permet de représenter les fonctions du type $T \to R$, alors que `BiFunction<T,U,R>` permet de représenter les fonctions du type $T \times U \to R$.
Complétez la méthode suivante pour qu'elle prennent en paramètre une fonction binaire du type $T \times U \to R$ et la convertisse en une fonction du type $T \to (U \to R)$

```
    <T, U, R> Function<T, Function<U, R>> curryfy(BiFunction<T, U, R> bf) {
        // a completer.
    }
```

Par example : pour la fonction qui additione deux nombres $(x, y) \to x + y$, la méthode `curryfy` doit retourner la fonction $x \to (y \to y + x)$.

Écrivez ici le code de la méthode `curryfy` :

```
return t -> u -> bf.apply(t, u);
```

ou (version typé)

```
return (T t) -> ((U u) -> bf.apply(t, u));
```

Question 12 (*)

```
public class C <T> { ..... }
public class D<T> extends C<T> { ..... }
```

Quelles sont, parmi les suivantes, les relations d'héritage qui sont valables ?

Réponses possibles :

→ `D<Double>` dérive de `C<Double>`

→ `D<Object>` dérive de `C<Object>`

× `D<Double>` dérive de `C<Object>`

× `C<Double>` dérive de `C<Object>`

Question 13 (*)

L'interface Java `Comparable` est utilisée pour ordonner les objets d'une classe définie par l'utilisateur. Elle est située dans le package `java.lang` et contient une seule méthode `public int compareTo(Object obj)` utilisée pour comparer l'objet courant avec l'objet spécifié. Elle retourne un entier positif si l'objet courant est le plus grand des deux, 0 s'ils sont égaux, et un nombre négatif si c'est l'argument qui est le plus grand.
Ecrire une méthode générique déterminant le plus grand élément d'un tableau, la comparaison des éléments utilisera l'ordre induit par la méthode `compareTo` de la classe des éléments du tableau.

Écrire ici les classes/interfaces demandées

```
import java.lang.Comparable; // inutile
```

Session 1 POOIG 2022/23

```
static <T extends Comparable<T> > T max (T[] valeurs){
 if (valeurs == null) return null ;
 if (valeurs.length == 0) return null ;
 T maxi = valeurs[0] ;
 for (T v : valeurs) if (v.compareTo(maxi) > 0) maxi = v ;
 return maxi ;
}
```

NE PAS RETOURNER AVANT D'EN ETRE AUTORISÉ