(Bon courage !)

Université Paris Cité

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

8 Janvier 2024

Durée : 2 heures

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 (*)

On considère les classes suivantes.

```
public class A {}
public class B extends A {}
public class C1 extends B {}
public class C2 extends B {}
```

Parmi les affirmations suivantes, lesquelles sont vraies ?

Réponses possibles :

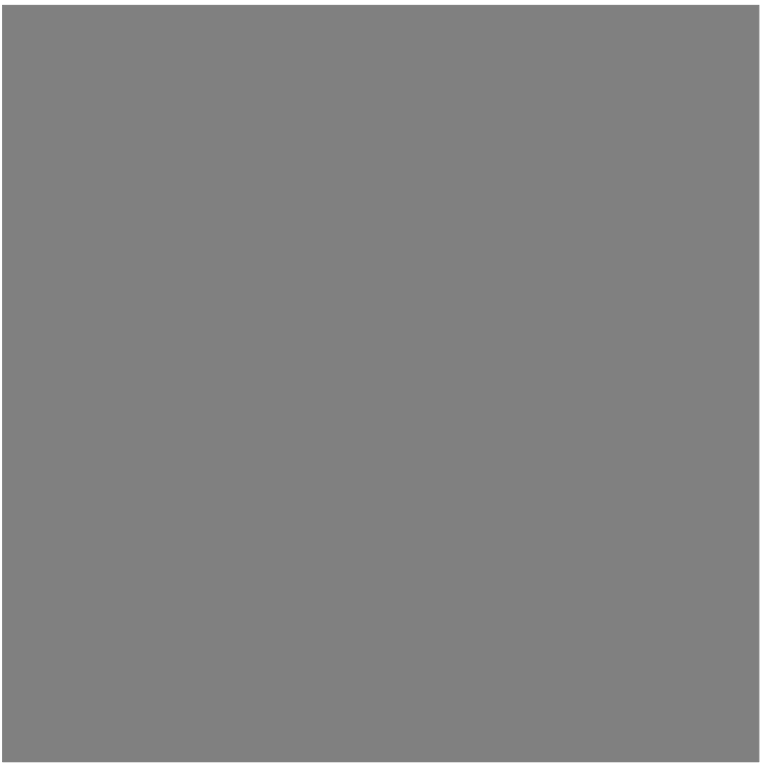
a. Le type `A` est un sous-type de `C1`. Faux, c'est l'inverse

b. Un objet de type `C1` ou `C2` est aussi de type `? extends B`. vrai

c. Un objet de type `C1` ou `C2` est aussi de type `? super B`. faux

d. Un objet de type `Object` est aussi de type `? super C1`. vrai

Session 1 POOIG 2023/24



Question 2 (**)

On considère le code suivant.

```
public class A {}
public class B extends A {}
public static Object f(List<Object> lst) {
  lst.add(new B()); return lst.get(0);
}
public static A g(List<B> lst) {
  lst.add(new B()); return lst.get(0);
}
public static <T extends A> T h(List<T> lst) {
  lst.add(new A()); return lst.get(0);
}
```

Parmi les affirmations suivantes, lesquelles sont vraies ?

Réponses possibles :

a. La méthode statique `f` compile sans erreur. vrai

b. La méthode statique `g` compile sans erreur. vrai

c. La méthode statique `h` compile sans erreur. faux, type inconnu

d. Une variable de type declaré `List<B>` peut être donnée en argument de `f`.
Faux, List<B> pas sous-type de List<A>

Question 3 (*)

Lesquelles de ces affirmations sont correctes ?

Réponses possibles :

a. Une classe peut toujours hériter de plusieurs classe en Java Faux

b. Une classe peut toujours implémenter plusieurs interfaces en Java Vrai

c. Une classe peut avoir au plus une classe ancêtre en Java Faux, à chercher

d. On peut implémenter une méthode dans une interface Vrai

Question 4 (*)

Voici le code suivant :

```java
//Dans le fichier Vehicule.java:
public class Vehicule {
    private int vitesse;
    public Vehicule(){
        this.vitesse = 0 ;
        System.out.println("A");
        }

    public void setVitesse ( int v ){
        this.vitesse = v;
    }
}
//Dans le fichier Urbain.java

public class Urbain extends Vehicule {
    private int nbRoues;
    public Urbain(int nbRoues){
        super();
        this.nbRoues = nbRoues;
        System.out.println("B");
    }
}
//Dans le fichier Test.java
public class Test {
    public static void main(String[] args){
        Vehicule v0 = new Vehicule();
        Urbain v1 = new Urbain();
     }
}
```

Comment se comporte le programme ?

Réponses possibles :

Ne compile pas, constructeur de urbain sans paramètre = erreur

a. Le programme affiche ”A”

b. Le programme affiche ”B”

c. Le programme affiche ”A” puis ”B”

d. Le programme ne compile pas

Question 5 (**)

On considère les classes suivantes.

```
public class A {
    public void m(A a){
        System.out.print("1");
    }
}

public class B extends A {
    public void m(A a){
        System.out.print("2");
    }
    public void m(B b){
        System.out.print("3");
    }
}
```

On déclare trois variables, `A a = new A()`, `B b = new B()` et `A c = new B()`. Qu'affiche la console si on exécute les trois intructions `a.m(b)`, `b.m(c)` et `c.m(b)` ?

Réponses possibles :

a. 123

b. 122

c. 132 <- car type déclaré qui compte et paramètre = type effectif

d. Un message d'erreur.

Question 6 (***)

On considère l'interface suivante :

```
public interface Fonction<T,S> {
    S evaluer(T t);
}
```

Écrivez en java une fonction `public ... dual(T t)` qui retourne la fonction qui prend en argument un objet `Fonction<T,S> f` et retourne `f.evaluer(t)`.

```
public static <T, S> Fonction <Fonction<T, S>, S> dual(T t){
    return new Fonction<Fonction<T, S>, S>(){
        @Override
        public S evaluer(Fonction<T, S> t){
            return t.evaluer(T);
} } }
```

Question 7 (**)

On considère les classes suivantes.

```
public class A {
  private class B {
    public int b1;
    protected int b2;
    private int b3;

    public B() {
```

```
      b1 = 1;
      b2 = 2;
      b3 = 3;
    }
  }

  public class C extends B {
    private int c4 = 0;
  }
}

class D {
  A a = new A();

  public D() {
    ...
  }
}
```

Qu'est-ce qu'on peut avoir au lieu de "..." dans le constructeur `D` pour que le programme compile ?

Réponses possibles :

a.
```
A.B b = a.new B();
int x = b.b3;
```
Non car B est privé dans la classe interne

b.
```
A.C c = a.new C();
int x = c.b2;
```
Oui

c.
```
A.C c = a.new C();
int x = c.b1;
```
Oui

d.
```
A.C c = a.new C();
int x = c.c4;
```
Non, c4 privé

Question 8 (**)

```
public class Test {
    public static void main(String[] args) {
        System.out.println(testMethod());
    }

    static int testMethod() {
        try {
            throw new Exception("A");
        } catch (Exception e) {
            return 1;
        } finally {
            return 2;
        }
    }
}
```

Qu'affiche le programme précédent ?

Réponses possibles :

a. `"A"`

b. 1 <- avec le catch

c. 2 <- avec le finally

d. Un message d'erreur.

Question 9 (*)

Soit le code suivant en Java :

```
class A {
  static int count = 0;
  A() {
    count++;
  }
}
class B extends A {
  B() {
    count++;
  }
}
```

Combien vaut `A.count` après la création d'un objet de type B (`new B()`) ? Supposer qu'il s'agisse du premier objet de classe `A` ou `B` créé dans le programme.

Réponses possibles :

a. 1

b. 2 <-

c. 0

d. Cela dépend de l'implémentation

Question 10 (**)

Considérez le programme suivant :

```
public class Test{
  public static int a = 0;
  public static void ex1(int x){ if(x==1) throw new RuntimeException(); }
  public static void ex2(int x){ if(x==2) throw new Exception(); }

  public static void f(int x){
    try{ex1(x);}
    catch(Exception e){a+=5;}
    finally{
      int[] tab = new int[10];
      for (int i = 0; i<30 ; i++)
        a += tab[i];
    }
  }
  public static void main(String [] args){
    try{ f(1);}
```

Déclarer l'exception dans la signature

Catch inatteignable

```
    catch(Exception e){a+=10;}
    catch(RuntimeException e){a+=10;}
    finally{a+=50; System.out.println(a);}
  }
}
```

1. Trouvez toutes les éventuelles erreurs de compilation de ce programme et expliquez pourquoi il s'agit d'une erreur.
2. Si on supprime les lignes contenant une erreur de compilation, qu'affiche le programme ?

65

Question 11 (**)

Considérons l'interface générique suivante :

```
public interface Predicate<T> {
  boolean test(T t)
}
```

Un objet `p` de type `Predicate<T>`, peut être vu comme un "prédicat" sur les instances d'une classe `T` quelconque, c'est-à-dire une condition vraie ou fausse selon la valeur de `t`. On dit que `t` *statisfait* le prédicat lorsque cette condition est vérifiée, c'est-à-dire lorsque `p.test(t)` renvoie `true`.

À l'aide de lambda-expressions, montrer comment créer deux objets `p` et `q` dont les classes sont des spécialisations de l'interface `Predicate`, et représentant respectivement les prédicats suivants :

- "`t` est une instance de `Integer` encapsulant entier non nul".
- "`t` est une instance de `String` représentant une chaîne non vide".

```
Predicate<Integer> p = x->x!=0
Predicate<String> q = s-> !s.isEmpty();
```

On souhaite ajouter à `Predicate` deux nouvelles méthodes avec une implémentation par défaut :

- `Predicate<T> not()`. Pour tout prédicat `p`, l'invocation `p.not()` doit renvoyer un nouveau prédicat représentant la négation de `p`.
- `Predicate<T> and(Predicate<T> q)`. Pour tous prédicats `p` et `q`, l'invocation `p.and(q)` doit renvoyer un nouveau prédicat représentant le "et" logique des prédicats `p` et `q`.

Donner le code de ces méthodes. Chacune devra renvoyer son résultat sous la forme d'une lambda-expression.

```
d
ef
a
ul
t
```

Question 12 (*)

On considère le code suivant

Session 1 POOIG 2023/24

```
interface I {
  public void f(I a);
  public void f(B n);
  public void f(A v);
}

class A implements I {
  public void f(I a) { System.out.print("A.f(I) ");}
  public void f(B n) { System.out.print("A.f(B) ");}
  public void f(A v) { System.out.print("A.f(A) ");}
}

public class B implements I {
  public void f(I a) { System.out.print("B.f(I) ");}
  public void f(B n) { System.out.print("B.f(B) ");}
  public void f(A v) { System.out.print("B.f(A) ");}
  public static void main(String args[]) {
    I a= new A();
    I b= new B();
    a.f(b);
    B a1 = new B();
    a1.f(a);
  }
}
```

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

Réponses possibles :

a. `A.f(I) B.f(B)`

b. `A.f(A) B.f(I)`

c. `A.f(I) B.f(I)` <-

d. `A.f(A) B.f(B)`

Page laissée vide intentionnellement

NE PAS RETOURNER AVANT D'EN ETRE AUTORISÉ