

Nom, prénom :

Contrôle de Compléments en Programmation Orientée Objet n° 1

Pour chaque case, inscrivez soit « V » (rai) soit « F » (aux), ou bien ne répondez pas.
Note = $\max(0, \text{nombre de bonnes réponse} - \text{nombre de mauvaises réponses})$, ramenée au barème.
Sauf mention contraire, les questions concernent Java 11.

Questions :

1. ☐ Quand, dans une méthode, on définit et initialise une nouvelle variable locale avec une instruction de la forme `Integer x = 12;`, alors la valeur 12 est stockée dans le tas.
2. ☐ Quand « `this` » apparaît dans une méthode, sa valeur est le récepteur de l'appel courant à celle-ci.
3. ☐ Toute classe possède au moins un constructeur.
4. ☐ `Number` est supertype de `double`.
5. ☐ Quand un objet n'est plus utilisé, il faut demander à la JVM de libérer la mémoire qu'il occupe.
6. ☐ La ligne 12 du programme ci-dessous affiche « 1 ».

```
1 class Truc {
2     static int v1 = 0; int v2 = 0;
3     public int getV1() { return v1; }
4     public int getV2() { return v2; }
5     public Truc() { v1++; v2++; }
6 }
7
8 public class Main {
9     public static void main(String args[]) {
10         System.out.println(new Truc().getV1());
11         System.out.println(new Truc().getV2());
12         System.out.println(new Truc().getV1());
13     }
14 }
```

7. ☐ La ligne 11 du programme ci-dessus affiche « 3 ».
8. ☐ Une interface peut avoir des instances directes.
9. ☐ Tout objet existant à l'exécution est instance de `Object`.
10. ☐ Le polymorphisme par sous-typage permet de réutiliser, dans un nouveau fichier `G.java` une méthode `f` définie dans le fichier `F.java` (sans recompilation de ce dernier) avec des paramètres effectifs dont le type n'avait pas encore été programmé quand `F.java` avait été compilé.
11. ☐ Il est plus facile de prouver qu'un programme se comporte correctement quand ses classes *encapsulent* leurs données que quand elles ne le font pas.
12. ☐ Une classe implémentant une interface `I` doit définir toutes les méthodes déclarées dans `I`.
13. ☐ La méthode `somme` ci-dessous s'exécute toujours normalement (sans lever `IndexOutOfBoundsException`) :

```
1 import java.util.List; import java.util.ArrayList;
2 public class PaquetDEntiers {
3     private final List<Integer> contenu; private final int taille;
4     public PaquetDEntiers(ArrayList<Integer> contenu) {
5         this.contenu = new ArrayList<>(contenu);
6         this.taille = this.contenu.size();
7     }
8     public int somme() {
9         int s = 0; for (int i = 0; i < taille; i++) { s += contenu.get(i); } return s;
10    }
11 }
```

14. ☐ Le patron de conception « adaptateur » consiste à écrire une classe implémentant une interface donnée, à l'aide d'une autre classe qui fournit les fonctionnalités de cette interface sans l'implémenter.

15. ☐ `javac` prend en entrée un code source Java et produit, en sortie, du code-octet.

16. ☐ Dans la classe suivante :

```
1 public class A {  
2     private int d;  
3     public A(int d) { this.d = d; }  
4     public int getD() { return d; }  
5 }
```

Pour s'assurer que l'appel à `getD` sur une même instance de `A` retourne toujours la même valeur, il est nécessaire d'ajouter, dans le constructeur, une copie profonde du paramètre `d`.

(On suppose que tout est exécuté sur le même thread.)

17. ☐ Quand on « `cast` » (transtype) une expression d'un type référence vers un autre, dans certains cas, Java doit, à l'exécution, modifier l'objet référencé pour le convertir.

18. ☐ Le système de sous-typage de Java est structurel (si une interface `T` possède toutes méthodes d'une autre interface `U`, avec des signatures compatibles, alors `T` est sous-type de `U`).

19. ☐ La classe d'un objet donné est connue et interrogeable à l'exécution.

20. ☐ Si `A` et `B` sont des types référence, `A` est sous-type de `B` si et seulement si toutes les instances de `A` sont aussi des instances de `B`.

21. ☐ Le type `byte` est primitif.

22. ☐ Le type `String` est primitif.

23. ☐ Tout seul, le fichier `A.java`, ci-dessous, compile :

```
1 public class A { final boolean a = 0; }  
2 class B extends A { final boolean a = 1; }
```

24. ☐ Dans la classe `B` ci-dessous, la méthode `f` de la classe `A` est surchargée par la méthode `f` de `B` :

```
1 class A { private static void f() {} }  
2 class B extends A { private static void f() {} }
```

25. ☐ Une classe peut avoir plusieurs sous-classes directes.

26. ☐ Pour les types référence, sous-typage implique héritage.

27. ☐ La dernière version de Java est Java 12.

28. ☐ Java est un langage orienté objet à prototypes.

29. ☐ La durée de vie d'un attribut non statique est celle d'une instance donnée de la classe.

30. ☐ Avec `x` et `y` de type `Object`, après exécution de l'instruction `x = y;`, la variable `x` représente désormais une copie de l'objet représenté par `y`.