Nom, prénom :

Partiel 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, nombre de bonnes réponse - nombre de mauvaises réponses), ramenée au barême.
Sauf mention contraire, les questions concernent Java 8.

1. ☐ Quand, dans une méthode, on définit et initialise une nouvelle variable locale de type `int`, sa valeur est stockée dans le tas.
2. ☐ Quand "`this`" est une expression, elle s'évalue comme l'objet sur lequel la méthode courante a été appelée.
3. ☐ Toute classe dispose d'un constructeur par défaut, sans paramètre.
4. ☐ `Object` est supertype de `double`.
5. ☐ Une variable locale est toujours déclarée à l'intérieur d'un bloc d'instructions.
6. ☐ On écrit `public` devant la déclaration d'une variable locale pour qu'elle soit visible depuis les autres classes.
7. ☐ Dès lors qu'un objet n'est plus utilisé, il faut penser à demander à Java de libérer la mémoire qu'il occupe.
8. ☐ La ligne 11 du programme ci-dessous affiche "2".

```
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 }
```

9. ☐ La ligne 12 du programme ci-dessus affiche "1".
10. ☐ La durée de vie d'un attribut statique est celle d'une instance donnée de la classe.
11. ☐ La plateforme Java est adaptée à la programmation système.
12. ☐ Dans un fichier source Java, une instruction se situe nécessairement (pas forcément seule) entre une accolade ouvrante et une accolade fermante.
13. ☐ 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`.
14. ☐ Une interface peut avoir des instances directes.
15. ☐ Tout objet existant a l'exécution est instance de `Object`.
16. ☐ Le polymorphisme par sous-typage permet de réutiliser une méthode `f(...)`, avec des paramètres effectifs de types différents entre deux utilisations, sans recompiler `f(...)`.
17. ☐ 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.
18. ☐ Les attributs d'une interface sont tous statiques.

19. ☐ Une classe implémentant une interface doit implémenter/redéfinir toutes les méthodes déclarées dans l'interface.
20. ☐ Pour faire un *downcasting*, on doit demander explicitement le transtypage.
21. ☐ La méthode `somme` ci-dessous s'exécute toujours sans erreur (exception) :

```java
import java.util.List; import java.util.ArrayList;
public class PaquetDEntiers {
    private final List<Integer> contenu; private final int taille;
    public PaquetDEntiers(ArrayList<Integer> contenu) {
        if (contenu != null) this.contenu = contenu;
        else this.contenu = Collections.emptyList(); // initialisation à liste vide
        this.taille = this.contenu.size();
    }
    public int somme() {
        int s = 0; for (int i = 0; i < taille; i++) { s += contenu.get(i); } return s;
    }
}
```

22. ☐ Java dispose d'un système de typage statique.
23. ☐ Le code source doit être compilé en code-octet avant chaque exécution.
24. ☐ Les objets sont typiquement stockés dans la pile.
25. ☐ Certaines vérifications de type ont lieu à l'exécution.
26. ☐ 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.
27. ☐ Un transtypage de référence se traduit toujours par une instruction spécifique dans le code-octet.
28. ☐ On peut déclarer une classe non imbriquée avec la visibilité `private`.
29. ☐ Dans une classe `B`, membre statique de `A` (on suppose que `B` ne contient pas elle-même de définition de type imbriqué), `this` désigne toujours une instance de `B`.
30. ☐ La classe d'un objet donné est connue et interrogeable à l'exécution.
31. ☐ La conversion de `int` vers `float` ne perd pas d'information.
32. ☐ Une interface définit un sous-type de `Object`.
33. ☐ Une interface peut avoir une classe membre.
34. ☐ Java est plus ancien que C++.
35. ☐ Le type d'une expression est calculé à l'exécution.
36. ☐ 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`.
37. ☐ Le type des objets Java est déterminé à l'exécution.
38. ☐ Le type `char` est primitif.
39. ☐ Le type `Object` est primitif.
40. ☐ La JVM interprète du code source Java.