Nom, prénom :

Partiel de Compléments en Programmation Orientée Objet n° 2

*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.

Rappel, quelques interfaces définies dans `java.util.function` :

```
public interface Consumer<T> { void accept(T t); }
```

```
public interface Function<T, R> { R accept(T t); }
```

Questions :

1. ☐ Les attributs d'une interface sont tous statiques.
2. ☐ Une classe implémentant une interface `I` doit (re)définir toutes les méthodes déclarées dans `I`.
3. ☐ La méthode `somme` ci-dessous s'exécute toujours sans erreur (ne quitte pas sur une exception) :

```
import java.util.List; import java.util.ArrayList; import java.util.Collections;
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;
    }
}
```

4. ☐ Le code source doit être recompilé en code-octet avant chaque exécution.
5. ☐ 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.
6. ☐ La conversion de `long` vers `double` ne perd pas d'information.
7. ☐ Une interface peut contenir une `enum` membre.
8. ☐ Le type des objets Java est déterminé à l'exécution.
9. ☐ Tout seul, le fichier `A.java`, ci-dessous, compile :

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

10. ☐ Une `enum` peut hériter d'une autre `enum`.
11. ☐ Une `enum` peut avoir plusieurs supertypes directs.
12. ☐ Quand, dans une classe, on définit une méthode de même nom qu'une méthode héritée, il y a nécessairement masquage ou redéfinition de cette dernière.
13. ☐ Le type de l'argument de la méthode `add` d'une instance donnée de `LinkedList` est connu et interrogeable à l'exécution.
14. ☐ `HashSet<Integer>` est sous-type de `Set<Integer>`.
15. ☐ `Deque<Integer>` est sous-type de `Deque<Object>`.

16. ☐ Le compilateur autorise à déclarer une `enum` qui soit sous-type de `Iterable<Boolean>`.
17. ☐ Une classe peut avoir plusieurs sous-classes directes.
18. ☐ Une classe `final` peut contenir une méthode `abstract`.
19. ☐ Une classe `abstract` peut contenir une méthode `final`.
20. ☐ Pour les types référence, sous-typage implique héritage.
21. ☐ Dans la classe `B` ci-dessous, la méthode `f` de la classe `A` est masquée par la méthode `f` de `B` :

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

22. ☐ Il est interdit de placer à la fois `private` et `abstract` devant une déclaration de méthode.
23. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile :

```
public class Z<T> {}
class W<Integer> extends Z<T> {}
```

24. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile (rappel : `Integer` est sous-classe de `Number`) :

```
public class Z<T extends Number> { static Z<Integer> w = new Z<>(); }
```

25. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile :

```
import java.util.function.*;
public class Z { Function<Object, Boolean> f = x -> { System.out.println(x); }; }
```

26. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile :

```
import java.util.function.*;
public class Z { Consumer<Object> f = System.out::println; }
```

27. ☐ Tout seul, le fichier `LC.java`, ci-dessous, compile et garantit que toute instance de `LC` jamais créée (sauf modification de la classe `LC`) sera toujours soit une instance `LC.Cons`, soit de `LC.Empty`.

```
public class LC {
    private LC() {}
    public static class Empty extends LC { private Empty() {} }
    public static class Cons extends LC {
        public final int head; public final LC tail;
        public Cons(int head, LC tail) { this.head = head; this.tail = tail; }
    }
    public static Empty empty = new Empty();
}
```

28. ☐ Même question que la précédente en remplaçant “instance” par “instance directe”.
29. ☐ Il est possible, depuis l’extérieur, de créer une instance de `LC.Empty` différente de `LC.empty`.
30. ☐ Dans le programme ci-dessous, le type `Livre` est immuable :

```
public class Livre {
    public final String titre, auteur;
    private Livre(String auteur, String titre) { this.auteur = auteur; this.titre = titre; }
    public static final class Roman extends Livre {
        public Roman(String auteur, String titre) { super(auteur, titre); }
    }
    public static final class Essai extends Livre {
        public Essai(String auteur, String titre) { super(auteur, titre); }
    }
}
```