Nom, prénom : **Exemplaire n° :** 9

Contrôle 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.

Questions : *(Sauf mention contraire, les questions concernent Java 11.)*

1. ☐ Le type d'une expression est calculé à l'exécution.
2. ☐ Quand, dans une méthode, on définit et initialise une nouvelle variable locale de type `int`, à l'exécution, sa valeur est stockée dans le tas.
3. ☐ Tout seul, le fichier `LC.java`, ci-dessous, compile et le type `LC` est scellé. [1]

```java
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();
}
```

4. ☐ Certaines vérifications de type ont lieu à l'exécution.
5. ☐ Les opérations sur un attribut `volatile` sont atomiques [2] (par rapport à celui-ci).
6. ☐ Placer le mot-clé `synchronized` devant toutes les méthodes d'une classe les rend atomiques [3] (par rapport aux attributs de cette classe).
7. ☐ Une `enum` peut avoir plusieurs supertypes directs.
8. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile :

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

Rappel : `public interface Function<T, R> { R apply(T t); }`

9. ☐ Les objets sont typiquement stockés dans la pile.
10. ☐ Une méthode ne peut pas être à la fois `private` et `abstract`.
11. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile :

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

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

12. ☐ Tout seul, le fichier `Z.java`, ci-dessous, compile (rappel : `Integer` étend `Number`) :

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

1. Type scellé : type `T` dont la liste des sous-types est connue et fixée définitivement à la compilation de `T`.
2. Voir 3.
3. Opération/Méthode atomique par rapport à un ensemble de variables : opération/méthode dont les accès, à cet ensemble de variables, effectués par une exécution de celle-ci ne sont pas entrelaçables avec les accès, à des variables du même ensemble, effectués par d'autres opérations ou méthodes.

13. ☐ Java dispose d'un système de typage statique.
14. ☐ Une classe **abstract** peut contenir une méthode **final**.
15. ☐ La durée de vie d'un attribut statique est celle d'une instance donnée de la classe.
16. ☐ `Object[]` est un supertype de `Integer[]`.
17. ☐ Le même programme sera exécuté environ 2 fois plus rapidement sur un ordinateur dont le CPU (microprocesseur) a 4 cœurs que sur un ordinateur dont le CPU en a 2 (toutes les autres caractéristiques du matériel restant identiques par ailleurs).
18. ☐ Dans le programme ci-dessous, le type `Livre` est immuable[4] :

```
1  public class Livre {
2      public final String titre, auteur;
3      private Livre(String auteur, String titre) { this.auteur = auteur; this.titre = titre; }
4      public static final class Roman extends Livre {
5          public Roman(String auteur, String titre) { super(auteur, titre); }
6      }
7      public static final class Essai extends Livre {
8          public Essai(String auteur, String titre) { super(auteur, titre); }
9      }
10 }
```

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

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

20. ☐ Toute classe dispose d'un constructeur sans paramètre.
21. ☐ Appeler la méthode `start` sur une instance de `Thread` démarre un nouveau *thread*.
22. ☐ Une classe **final** peut contenir une méthode **abstract**.
23. ☐ Les variables locales peuvent être des variables partagées (entre *threads*).
24. ☐ La JVM interprète du code source Java.
25. ☐ `HashSet<Integer>` est sous-type de `Set<Integer>`.
26. ☐ Une **enum** peut hériter d'une autre **enum**.
27. ☐ `Deque<Integer>` est sous-type de `Deque<Object>`.
28. ☐ L'instruction ci-dessous a pour effet d'afficher : `1 1`.[5]

```
1     Stream.of(1,2,3).peek(x -> System.out.print(x + " ")).limit(1).forEach(System.out::print)
```

29. ☐ Dans la classe `B` ci-dessous, la méthode `f` définie dans `B` masque la méthode `f` héritée :

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

30. ☐ Le mot-clé **volatile** devant un attribut empêche les accès en compétition à celui-ci.

4. Type immuable : type dont les instances directes ou indirectes, présentes ou futures sont non modifiables.
5. La méthode `Stream<T> peek(Consumer<T> ope)` de `Stream<T>` est une opération intermédiaire retournant un *stream* identique à **this**, mais dont le traitement exécutera `ope` sur chaque élément du *stream*.
La méthode `Stream<T> limit(int n)` est une opération intermédiaire de `Stream<T>` retournant un *stream* identique à **this** mais tronqué aux `n` premiers éléments.
La méthode **void** `forEach(Consumer<T> ope)` est une opération terminale de `Stream<T>` qui exécute `ope` sur chaque élément du *stream* sur lequel elle a été appelée.