De Java 11 à Java 17, les nouveautés

Tu vas avoir toutes les modifications qui vont te changer la vie de tous les jours sous les yeux en une seule fois !

Apprendre les nouveautés de Java 11 à Java 17

Pourquoi une fiche sur les nouveautés de Java 11 à Java 17 ? Parce que c'est une période de transition pour beaucoup de développeurs.

En effet, les chiffres changent, mais les faits restent. En 2022, à la publication de cette fiche, New Relic publiait un rapport sur l'usage de Java :

	Java 8	**Java 11**
2020	84,48 %	11,11%
2022	46,45 %	48%

Comme Java 17 est la nouvelle version LTS de Java, il y a fort à parier que les migrations seront nombreuses. De plus, les versions 12 à 16 sont assez peu utilisées. J'ai décidé de faire le grand écart, comme beaucoup et de passer de la 11 à la 17. Oui, cette fiche servira donc au moins à une personne !

La version 17 ne sera rapidement plus la dernière version LTS, mais les nouveautés de Java 17 sont toujours d'actualité. De plus, sa période de maintenance court jusqu'en 2026 et son support étendu jusqu'en 2029.

De plus comme les nouveautés présentées ici ne sont que des améliorations de syntaxe ou des ajouts de fonctionnalités mineures, elles sont souvent ignorées par les développeurs. Pourtant, elles peuvent te changer la vie de tous les jours.

La fiche technique

Switch Case

Tu peux cumuler toutes ces fonctionnalités entre elles !

- `case` multiple avec un séparateur `,` @since Java 14

```
switch (i) {
  case 1, 2, 3 :
    System.out.println("Value is " + i);
    break;
  default :
    throw new IllegalArgumentException("Number is not
        supported");
}
```

- `->` remplace `:` et évite le `break` @since Java 12

```
switch (i) {
  case 1, 2, 3 -> System.out.println("Value is " + i);
  default      -> throw new IllegalArgumentException("Number
        is not supported");
}
```

- `->` peut être utilisé pour renvoyer une valeur @since Java 14

```
String vehicleType = switch (wheelNumber) {
  case 2  -> "bicycle";
  case 4  -> "car";
  default -> "Unknown vehicle";
};
```

- `->` switch sur le type pour caster à la volée @since Java 17 preview

```
static String getType(Object o) {
  return switch (o) {
    case Integer i -> String.format("%d is an integer", i);
    case Long l    -> String.format("%d is a big integer!",
        l);
    case Double d  -> String.format("%f is a decimal
        number!", d);
    case String s  -> String.format("%s is a String!", s);
    default        -> "Unknown type";
  };
}
```

- gestion du `null` @since Java 17 preview

```
switch (o) {
  case null  -> String.format("%d is an integer", i);
  case "you" -> System.out.println("pi !");
  case "hou" -> System.out.println("rra !");
  default    -> System.out.println("ouin...");
};
```

- `yield` pour renvoyer une valeur dans le switch sur le type @since Java 17 preview

```
static void toBool(int i) {
  Boolean b =  switch (i) {
    case 0 -> {
      System.out.println("c'est rien");
      yield false;
    }
    case 1 -> {
      System.out.println("c'est le début");
      yield true;
    }
    default -> {
      System.out.println("ce n'est pas possible");
      yield null;
    }
  };
}
```

Class modifier

- `record` : nouveau type de class : immutable et génère `getter/hashCode/equals` @since Java 14

```
record ColorRecord(int red, int green, int blue) {};
ColorRecord color = new ColorRecord(255, 10, 10);
color.red();
```

- `sealed` : afin de verrouiller les possibilités d'héritage @since Java 16

```
abstract sealed class Person permits Employee, Boss {
  String name;
}
final class Employee extends Person {
  int id;
  int getEmployeeId() {
    return id;
  }
}
final class Boss extends Person { }

// Va générer une erreur à la compilation
final class Customer extends Person { }
```

Formatage

- `NumberFormat` : ajout d'un formatage `SHORT` : 10k, 1M, etc. @since Java 12

```
NumberFormat.getCompactNumberInstance(Locale.FRANCE,
        NumberFormat.Style.SHORT);
```

- `NumberFormat` : ajout d'un formatage `LONG`; avec `Locale` : 1 million, 2 mille, etc. @since Java 14

```
NumberFormat.getCompactNumberInstance(Locale.FRANCE,
        NumberFormat.Style.LONG);
```

- `NumberFormat` : ajout d'un formatage monnaie, avec `Locale` : $3.24, 5,61€, etc. @since Java 14

```
NumberFormat.getCurrencyInstance(Locale.FRANCE)
```

- `DateTimeFormatter` : ajout du cycle, avec `Locale` : du matin, de l'après-midi, etc. @since Java 16

```
DateTimeFormatter.ofPattern("B", Locale.FRANCE)
```

Divers

- `String` : bloc String multiligne avec `"""` @since Java 13

```
// pour le même rendu :
String withoutMultiline = "{\n" +
    "  \"id\":213,\n" +
    "  \"name\":\"Nathaniel\"\n" +
    "}";
String withMultiline = """
    {
      "id":213,
      "name":"Nathaniel"
    }""";
```

- `Exception` : meilleur message de debug sur les `Exception`, en particulier les `NullpointerException` @since Java 17

```
String s = null;
System.out.println(s.toLowerCase());
// => Exception in thread "main"
        java.lang.NullPointerException: Cannot invoke
        "String.toLowerCase()" because "s" is null
```

- `instanceof` : peut être suivi d'un nom de variable afin de caster immédiatement @since Java 16

```
Person p = new Employee();
if (p instanceof Employee employee){
    System.out.println(employee.getEmployeeId());
}
```

- `Stream.toList` : afin d'éviter de passer par un `.collect(Collectors.toList())` @since Java 16

```
// avant
List.of("some","thing").stream().collect(Collectors.toList());
// maintenant
List.of("some","thing").stream().toList();
```

Ressources

Sources

Comme indiqué au fur et à mesure, les Release Notes des différentes versions de Java.

Fiches à télécharger

Tu peux télécharger ce devento en plusieurs formats : - pdf - md

---