

JAVA 24 QUOI DE NEUF ?



Java™



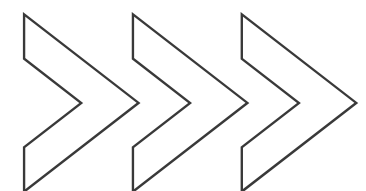
1. Langage : Écrire moins, faire plus

- **JEP 488 : Patterns avec types primitifs**

```
1 Object value = 42;
2 if (value instanceof int i) { // ✅ Autorise les primitifs
3     System.out.println(i * 2); // 84
4 }
5
6 // Switch avec double :
7 double temperature = 25.5;
8 String alert = switch (temperature) {
9     case Double d when d > 30.0 -> "Canicule !";
10    case Double d when d < 0.0  -> "Gel !";
11    default -> "OK";
12 };
13
```

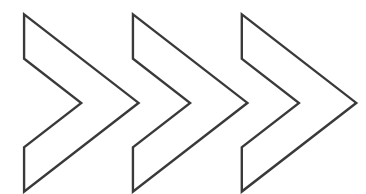
- **JEP 492 : Constructeurs flexibles**

```
1 class Robot {
2     String model;
3     Robot(String serial) {
4         // Prologue : logique avant super()
5         String prefix = "ROB-" + serial.substring(0, 3);
6         super(prefix); // Appel explicite possible après prologue
7         // Épilogue : initialisation finale
8         this.model = "X2000";
9     }
10 }
11
```



- **JEP 495 : Méthode main simplifiée**

```
1 void main() { // Pas besoin de `static` !  
2     System.out.println("Hello Java 24 !");  
3 }  
4
```



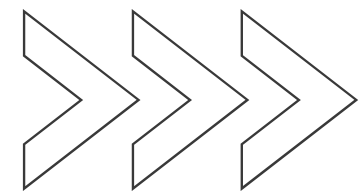
2. Bibliothèques : Productivité Boost

- **JEP 489 : API Vectorielle (calculs haute performance)**

```
1 VectorSpecies<Float> SPECIES = FloatVector.SPECIES_256;
2 float[] a = new float[1000], b = new float[1000], c = new float[1000];
3
4 for (int i = 0; i < a.length; i += SPECIES.length()) {
5     FloatVector va = FloatVector.fromArray(SPECIES, a, i);
6     FloatVector vb = FloatVector.fromArray(SPECIES, b, i);
7     va.mul(vb).intoArray(c, i); // ➡ x8 plus rapide qu'une boucle classique !
8 }
9
10
```

- **JEP 499 : Concurrency Structurée**

```
1 try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
2     Future<String> user = scope.fork(() -> fetchUser());
3     Future<String> order = scope.fork(() -> fetchOrder());
4     scope.join();
5     System.out.println(user.get() + " | " + order.get()); // ✅ Géré comme une unité
6 }
7
8
```



3. Sécurité : Préparation Post-Quantique

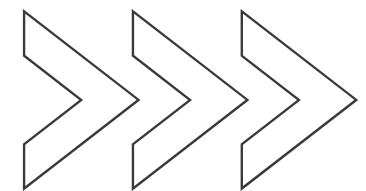
- **JEP 496/497 : Cryptographie Résistante aux Ordinateurs Quantiques**

```
1 KeyPairGenerator kpg = KeyPairGenerator.getInstance("ML-DSA");
2 KeyPair keyPair = kpg.generateKeyPair();
3 Signature sig = Signature.getInstance("ML-DSA");
4 sig.initSign(keyPair.getPrivate());
5 sig.update(message);
6 byte[] signature = sig.sign(); // 🛡️ Sécurisé contre les attaques futures
7
8
```

4. Outils : Optimisation des Builds

- **JEP 493 : Runtime sans JMOD**

```
1 jlink --add-modules java.base --output mon_jre_mini
2 # ➡ Taille réduite de 25% !
3
4
```



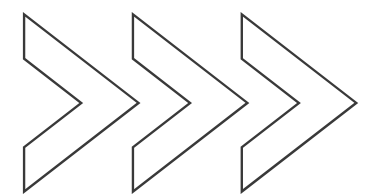
5. Performances : Rapidité et Mémoire

- **JEP 450 : En-têtes d'objets plus petits**

Résultat : Les objets prennent moins de mémoire (ex: new String() gagne 32 bits).

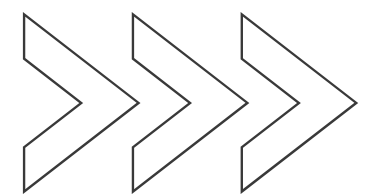
- **JEP 491 : Threads virtuels optimisés**

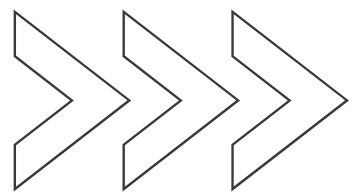
```
1 synchronized (lock) {  
2     // ✅ Plus de blocage des threads physiques !  
3     Thread.sleep(1000); // Libère le thread sous-jacent  
4 }  
5
```



Récap'

- **Productivité** : Moins de code pour les mêmes fonctionnalités.
- **Performance** : Optimisations mémoire et calcul vectoriel.
- **Avenir** : Sécurité prête pour l'ère quantique.





Merci