
# Formation Java 8 à 17

## Introduction

# Sommaire


Méthodes dans les interfaces

Méthodes de création de collections

Inférence de type

Date & Time API

Expression Lambda

API Stream

Fonctions du JDK

Pattern Optional

Les blocs de texte

Les records

Les classes sealed


# Tour d’horizon

# Introduction

| Date | Version | Caractéristiques |
|---|---|---|
| **2014** | **8** | Nouvelle API Date, du code dans les interfaces, les expressions lambdas et les streams pour traiter les collections de manière "fonctionnelle".<br>Version LTS |
| **09/2017** | **9** | Instauration des modules (niveau de visibilité supplémentaire)<br>Support de HTTP2 |
| **03/2018** | **10** | Inférence de type<br>var liste = new ArrayList<String>(); |
| **09/2018** | **11** | Nouveau garbage collector : ZGC (garbage collector à faible latence)<br>Version LTS |
| **03/2019** | **12** | Nouvelle syntaxe pour les switch<br>Nouveau Garbage Collector : Shenondoah aka G1GC. |
| **09/2019** | **13** | Text blocks avec le guillemet triple |
| **03/2020** | **14** | Version améliorée du instanceof |
| **09/2020** | **15** | Classes sealed en beta |
| **03/2021** | **16** | Types record en beta<br>Améliorations sur la VM et les GC |
| **09/2021** | **17** | Améliorations sur le gestion des threads et sur les GC<br>Les Classes et interfaces sealed et les types record officiellement publiés. |
| **03/2022** | **18** | Nouvelle API pour le calcul vectoriel<br>Un serveur web minimaliste intégré<br>Amélioration de l'API java.lang.reflect |

# Gains de performance

Dépend du Garbage Collector utilisé.

Au total 5 GC disponibles dans la version 17 :

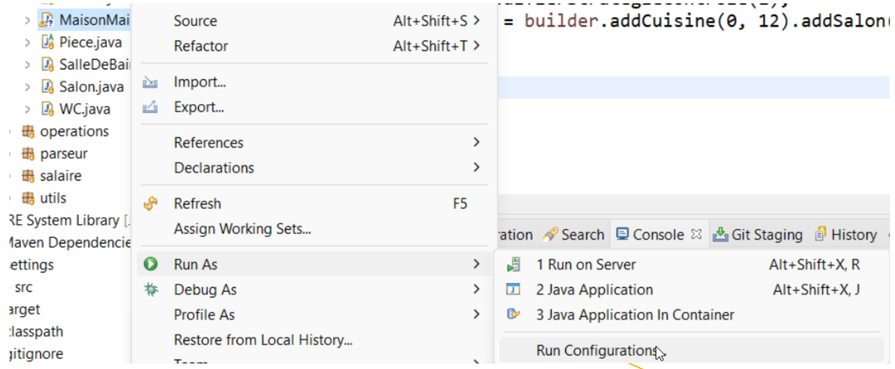
| Nom | Commentaires | Utilisation du GC |
|---|---|---|
| Serial GC | Garbage Collector mono-thread qui fonctionne bien sur les machines avec un processeur unique et plutôt peu de mémoire utilisée par l'application. A éviter sinon.<br>Grosse latence. | -XX:+UseSerialGC |
| Parallel GC | Appelé aussi Throughtput Collector (Throughput = Débit)<br>Utilise plusieurs threads en //.<br>Ne fonctionne que sur des machines avec plusieurs processeurs.<br>A privilégier pour du traitement de données sous forme de batchs par exemple.<br>Grosse latence. | -XX:+UseParallelGC |
| G1 GC | Garbage-First collector. Apparu dans la version Java 7.<br>Bon GC multirôle même si plutôt conçu pour des applications avec beaucoup de mémoire (au dessus de 6Go avec 50% d'espace occupé).<br>**GC par défaut.** | XX:+UseG1GC |
| Z GC | Expérimental.<br>Convient pour des applications nécessitant des mémoires de plusieurs To.<br>Faible latence: 10ms de pause max. | -XX:+UseZGC |
| Shenandoah | Expérimental.<br>GC à faible latence quelle que soit la taille mémoire occupée. | -XX:+UseShenandoahGC |

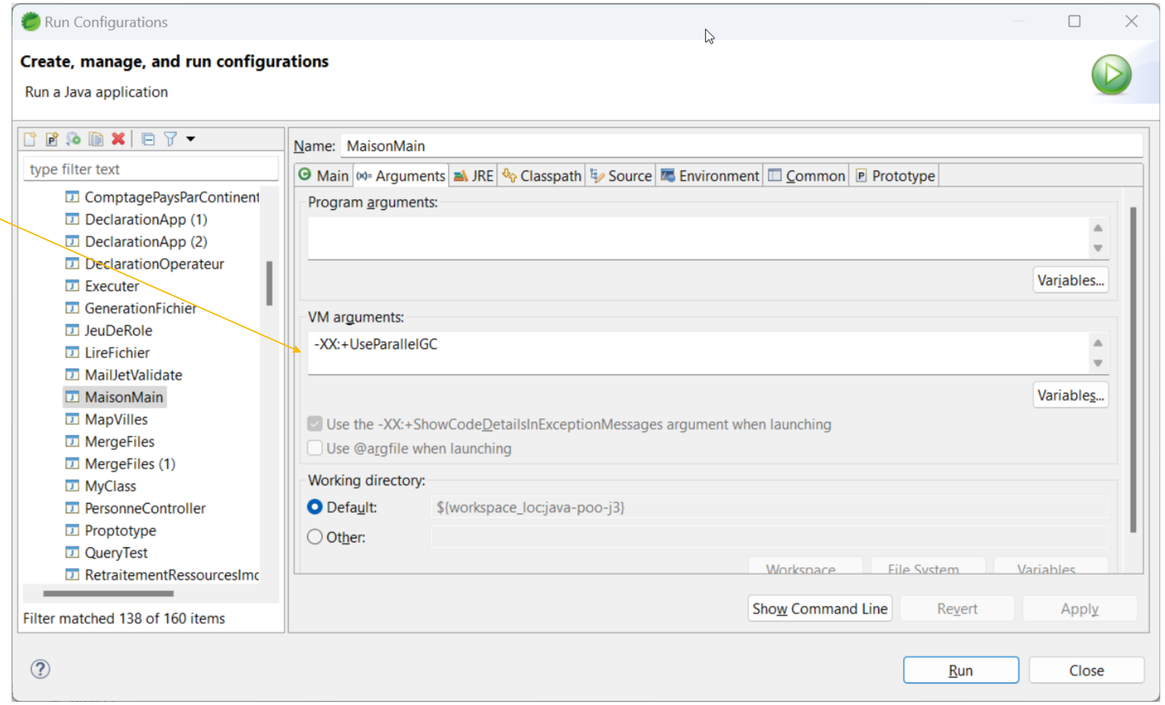
# Gains de performance – les différents GC

Java 17 est 8,66% plus rapide que Java 11 avec G1 GC

Java 17 est 6,54% plus rapide que Java 11 avec Parallel GC

# Pour changer de GC sur Eclipse



# Tour d’horizon

# Des méthodes dans les interfaces

```
interface MyInterface {

    public default void myMethod() {
        System.out.println("Je peux contenir du code");
        myPrivateMethod();
    }

    private void myPrivateMethod() {
        System.out.println("Je peux aussi contenir du code");
    }

    public static void myStaticMethod() {
        System.out.println("Je peux également contenir du code");
    }
}
```

# De nouvelles méthodes pour créer des collections

```
List<String> listImmutable = List.of("1", "2", "3", "4");

Set<String> setImmutable = Set.of("1", "2", "3", "4");
```

# Inférence de type

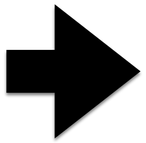
```
List<String> list = List.of("Diginamic", "aime", "Java");

// Peut aussi s'écrire
var otherList = List.of("Diginamic", "aime", "Java");
```

# Date & Time API

~~java.util.Date~~

~~java.util.Calendar~~



**java.time.LocalDate**

**java.time.LocalTime**

**java.time.LocalDateTime**

**java.time.Instant**

**java.time.Period**

**java.time.Duration**

# Expression Lambda

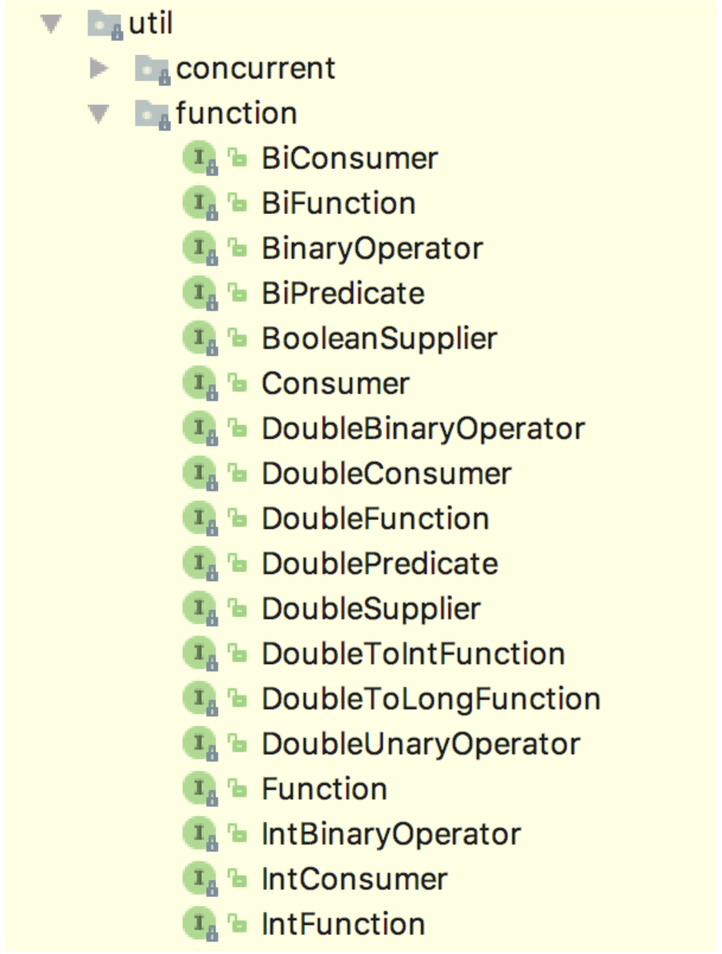
**pizza -> pizza.getPrice()**

# API Stream – approche fonctionnelle

```
listeCommandes.stream()
                .filter(x -> x.numero.startsWith("201705")
                .map(x -> x.montant*0.2)
                .sorted()
                .collect(Collectors.toList());
```

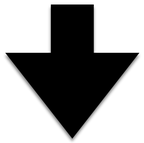
# Fonctions du JDK

```
package java.util.function;
```



# Pattern Optional

~~**NullPointerException**~~



**Optional<T>**

# Les blocs de texte

## Opérateur """

```
String json = "{\n\"nom\":\"Durand\",\n"
            + "\"prenom:\"Pauline\"\n"
            + "}";
System.out.println(json);
```

```
{
"nom":"Durand",
"prenom:"Pauline"
}
```

```
String json2 = """
               {
               "nom":"Durand",
               "prenom:"Pauline"
               }
               """;
System.out.println(json2);
```

```
{
"nom":"Durand",
"prenom:"Pauline"
}
```

# Les records - avant

**Avant**

1 classe avec 2 attributs

Un contructeur

Les getters/setters

Les méthodes equals et toString

**Beaucoup de code !!**

```
public class Individu {

   private String prenom;
   private String nom;

   public Individu(String prenom, String nom) {
      this.prenom = prenom;
      this.nom = nom;
   }

   @Override
   public boolean equals(Object obj) {
      if (!(obj instanceof Individu))
         return false;
      Individu other = (Individu) obj;
      return Objects.equals(nom, other.nom) && Objects.equals(prenom, other.prenom);
   }

   @Override
   public String toString() {
      return "Individu [prenom=" + prenom + ", nom=" + nom + "]";
   }

   + GETTERS (pas de setters)
}
```

# Les records – java 17

```
public record InvididuRecord(String nom, String prenom) {

}
```

**Les records**

- Mot clé record
- Les attributs d'instance sont déclarés dans la signature.
- Constructeur, equals, toString, getters et setters implicites

**Limitations :**

- Un record ne peut pas hérité d'une classe mère
- Un record est final, i.e. il ne peut pas avoir de classes filles

# Les classes Sealed

**Les classes Sealed connaissent leurs classes filles.**

- Mot clé **sealed** pour déclarer que la classe est scellée.
- Mot clé **permits** pour lister les classes filles.

**Contraintes :**

- Une classe fille doit être déclarée **non-sealed**, **final** ou **sealed**
- Une classe non-sealed peut avoir des classes filles

```
public sealed class Saison permits Printemps, Ete, Automne, Hiver {

}
```

```
public non-sealed class Ete extends Saison {

}
```

```
public final class Hiver extends Saison {

}
```