

ALOM

MODERN JAVA - DE JAVA 5 À 23

JAVA, UN LANGAGE VIEILLISSANT

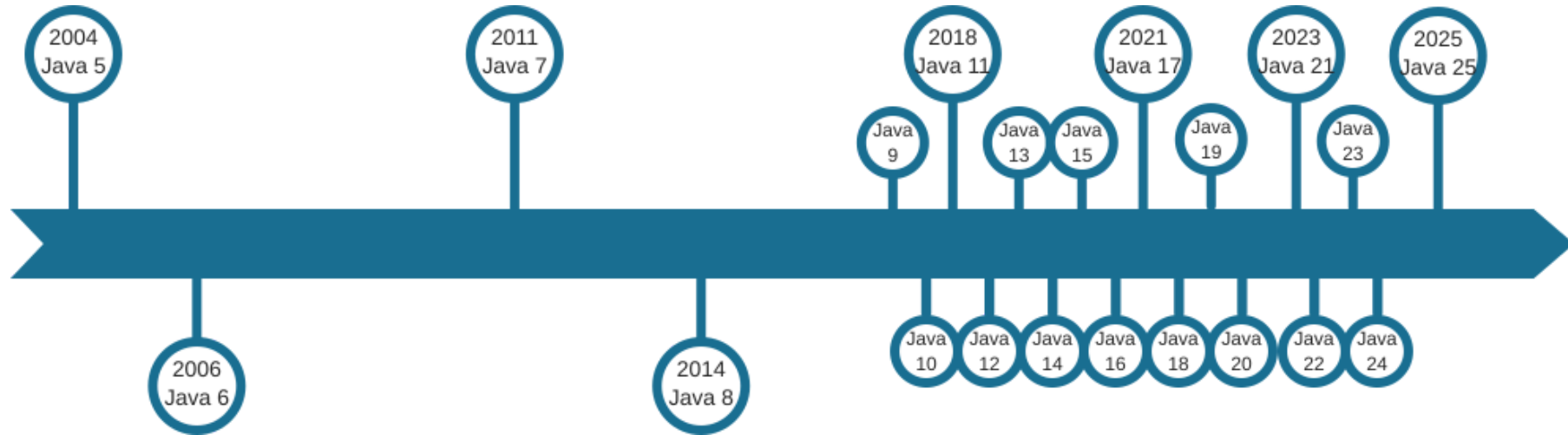


- Un cycle de release long (5 ans entre Java 6 et 7)
- Perte de vitesse face aux nouveaux langages émergeants
 - Kotlin
 - Javascript/TypeScript
 - Go
- En 2017 Oracle décide d'accélérer le développement de Java à partir de Java 9

NE SOYEZ PAS COMME KENNY



JAVA VERSIONS TIMELINE



Java 11 a déjà 7 ans 🧒 🎂

Java 17 a déjà 4 an 🧒 🎂

Java 21 a déjà 2 an 🧒 🎂

Il serait temps de s'y mettre !

PLAN

Ce cours liste les nouveautés de Java 5 à Java 23

Certaines versions n'apportent que peu de changement pour les développeurs :

- Ajout/suppression d'algorithmes de garbage-collection
- Fonctionnalités bas niveau

Ces versions ne sont pas listées dans ce cours

JAVA 5 - SEP 2004

- Types génériques : `Collection<T>`
- for each : `for (Car c: cars){}`
- Autoboxing des types primitifs

JAVA 7 - JUL 2011

DIAMOND OPERATOR FOR GENERICS :

```
Map<String , List <Trade>> trades = new TreeMap <> ();
```

SWITCH STRING

```
boolean isWeekEnd = false;
switch(day){
    case "SATURDAY":
    case "SUNDAY":
        isWeekEnd = true;
        break;
    default:
        break;
}
```

MULTI CATCH

```
try{  
    // code that throws exceptions  
}  
catch(IOException|SQLException e){  
    // do something with those exceptions  
}
```

AutoCloseable INTERFACE AND TRY-WITH-RESOURCE:

```
try( FileOutputStream fos = new FileOutputStream("text.txt") )  
    fos.write("Hello".getBytes());  
}  
catch(Exception e){  
    // do something with those exceptions  
}
```


JAVA 8 - MAR 2014

EXPRESSIONS LAMBIDAS

Permet d'implémenter des classes anonymes simplement.

Ex, avec l'instanciation d'un Runnable

```
Runnable runnable = new Runnable(){  
    @Override  
    public void run(){  
        System.out.println("Hello world !");  
    }  
};
```

Avec une lambda:

```
Runnable runnable = () -> System.out.println("Hello world !");
```



LAMBDA & INTERFACES FONCTIONNELLES

interface ne contenant qu'une seule méthode abstraite.

- `Comparator<T>` définit la méthode `int compare(T o1, T o2)`
- `Callable<V>` définit la méthode `V call()`
- `Supplier<T>` définit la méthode `T get()`
- `Runnable` définit la méthode `run()`
- `Consumer<T>` définit la méthode `accept(T t)`
- et beaucoup d'autres dans le JDK

Exemple avec Collections.sort:

```
List<Pokemon> pokemons = Arrays.asList(
    new Pokemon("Pikachu", 25),
    new Pokemon("Bulbizarre", 1)
);

Collections.sort(pokemons, new Comparator<Pokemon>() {
    @Override
    public int compare(Pokemon p1, Pokemon p2) {
        return p1.getName().compareTo(p2.getName());
    }
});
```

Exemple avec une lambda:

```
List<Pokemon> pokemons = Arrays.asList(
    new Pokemon("Pikachu", 25),
    new Pokemon("Bulbizarre", 1)
);

Collections.sort(pokemons, (p1, p2) -> p1.getName().compareTo(p2.getName()));
```

Instanciación de Lambdas

```
Runnable monTraitement = () -> System.out.println("traitement")

Consumer<String> afficher = param -> System.out.println(param)

Comparator<Pokemon> comp =
    (Pokemon p1, Pokemon p2) -> p1.getName().compareTo(p2.getN

Function<Integer, Boolean> isPositive = it -> it > 0;

Supplier<Integer> fortyTwo = () -> 42;
```

Syntaxe d'une lambda:

`() -> expression`

`() -> { traitements; }`

`(paramètres) -> expression`

`(paramètres) -> { traitements; }`

`paramètre -> expression`

METHOD REFERENCES

Les références de méthodes permettent d'invoquer une méthode existante comme une lambda:

- une méthode statique
- une méthode d'instance
- un constructeur

Syntaxe:

```
nomClasse::nomMethode ou  
objet::nomMethode
```

METHOD REFERENCES

référence de méthode	lambda
<code>System.out::println</code>	<code>x -> System.out.println(x)</code>
<code>Math::pow</code>	<code>(x, y) -> Math.pow(x, y)</code>
<code>pokemon::attack</code>	<code>x -> x.attack(x)</code>
<code>Pokemon::new</code>	<code>(x, y) -> new Pokemon(x, y)</code>

COLLECTIONS ET STREAMS

Les streams facilitent le traitement de données sur les collections.

- boucles sur les collections
- approche fonctionnelle
 - utilisation de Lambdas

COLLECTIONS ET STREAMS

Opérations sur les streams:

- filter : filtre du stream
- map : transformation
- find : recherche
- sorted : tri
- match : correspondance avec un prédicat

COLLECTIONS ET STREAMS

Réductions:

- count
- limit
- min / max
- reduce

Consommation:

- forEach
- toArray / toList
- collect

COLLECTIONS ET STREAMS

```
int countElectricLegendaries = pokemons.stream()  
    .filter(p -> p.getType().equals("electric"))  
    .filter(p -> p.isLegendary())  
    .count(); // 1
```

```
Set<String> legendaryTypes = pokemons.stream()  
    .filter(Pokemon::isLegendary)  
    .map(Pokemon::getType)  
    .map(String::toUpperCase)  
    .collect(Collectors.toSet()); // ELECTRIC, ICE, FIRE,
```

DEFAULT METHODS IN INTERFACES

```
public interface PokemonPrinter{  
    default void print(Pokemon p){  
        System.out.println("Pokemon name: " + p.getName());  
    }  
}
```

Optional<T>

Classe permettant d'exprimer une valeur ou `null`.

Peut simplifier l'écriture de code, et évite l'utilisation de `null`

```
// pour créer un optional vide  
Optional.empty();  
  
// pour créer un optional contenant une valeur  
Optional.of("Some Value here");
```

Manipulation:

```
Optional<String> res = someMethod();  
// retourne la valeur ou une valeur par défaut  
return res.orElse("default value");
```

```
Optional<String> res = someMethod();  
res.ifPresent(value -> {  
    // permet de faire quelque chose si une valeur est présent  
});
```

```
Optional<String> res = someMethod();  
return res.orElseThrow(new RuntimeException("Pas de valeur !"))
```

API DATE/TIME

Depuis Java 1.0 ( jan 1996): `java.util.Date`

- Contient une date *et* une heure, sur le fuseau horaire de la machine

Depuis Java 8 ( mar 2014)

- package `java.time`
 - `LocalDate` : Date sans heure ni fuseau
 - `LocalDateTime` : Date avec heure sans fuseau
 - `ZonedDateTime` : Date avec heure et fuseau

JAVA 9 - SEP 2017

Introduit l'Open JDK et le nouveau cycle de release
(tous les 6 mois)

FONCTIONNALITÉS

SYSTÈME DE MODULES

JSHELL (REPL)

JAVA 10 - MAR 2018

INFÉRENCE DE TYPE LOCALE

Mot clé `var`. Fonctionne dans les méthodes (pas en attribut de classe). Nécessite une initialisation immédiate (appel de constructeur ou retour de méthode).

```
@Test
public void var_example() {
    var message = "Hello, Java 10";
    assertTrue(message instanceof String);
}
```

CONTAINER AWARENESS

Java s'adapte à l'exécution en containers 

JAVA 11 - SEP 2018

Version "LTS", supportée par Oracle jusque Septembre
2023 

NOUVEAU CLIENT HTTP

```
HttpClient httpClient = HttpClient.newBuilder().build();
HttpRequest httpRequest = HttpRequest.newBuilder()
    .GET()
    .uri(URI.create("http://localhost:" + port))
    .build();
HttpResponse httpResponse = httpClient.send(httpRequest, HttpR
assertThat(httpResponse.body()).isEqualTo("Hello from the serv
```

EXECUTABLE .JAVA FILES

`java` sait maintenant exécuter des fichiers sans avoir besoin de les compiler avant. La JVM compile le code en mémoire et exécute le premier `main` trouvé.

```
$ java HelloWorld.java  
Hello Java 11!
```

SUPPRESSION DES PACKAGES JAVA EE ET CORBA

Les packages suivant ont été supprimés en Java 11, ce qui peut occasionner des `ClassNotFoundException` si du code l'utilisait:

- `java.xml.ws`
- `java.xml.bind`
- `java.activation`
- `java.xml.ws.annotation`
- `java.corba`
- `java.transaction`



Ces packages ont été déplacés dans le projet 'Jakarta':

NOUVELLES MÉTHODES

`String::isBlank`, `Files.writeString`,
`Files.readString`, `Collection.toArray`

SUPPORT DE TLS 1.3

JAVA 14 - MAR 2020

SWITCH EXPRESSIONS

On peut allouer le résultat d'un switch à une variable:

```
var isWeekEnd = switch (day) {  
    case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"  
    case "SATURDAY", "SUNDAY" -> true;  
    default -> throw new IllegalArgumentException("What's a "  
};
```

NULLPOINTEREXCEPTIONS AMÉLIORÉES

Les exception indiquent quelle variable est nulle:

```
java.lang.NullPointerException: Cannot store to int array beca
```



JAVA 15 -  SEP 2020

TEXT BLOCKS

Chaines de caractères multilignes. Utilisation de `"""` pour délimiter un bloc.

```
var str = """
    A quick brown fox jumps over a lazy dog;
    the lazy dog howls loudly.
    """;
System.out.println(str);
```

L'indentation est conservée par rapport aux `"""` fermants.

```
A quick brown fox jumps over a lazy dog;
    the lazy dog howls loudly.
```

JAVA 16 - MAR 2021

Le code de Java est maintenant sur Github.

Stream.toList

Simplifie le code des streams:

```
List<Integer> ints = integersAsString
    .stream()
    .map(Integer::parseInt)
    .collect(Collectors.toList());

List<Integer> intsEquivalent = integersAsString
    .stream()
    .map(Integer::parseInt)
    .toList();
```


RECORDS

Type de classe, immutable:

```
public class Person {  
    private final String name;  
    private final int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
}
```

Le constructeur principal ("cannonique") est déclaré dans l'en-tête du record.

RECORDS

```
public record Person(String name, int age) {  
}
```

Les champs sont en lecture-seule, pas de setter.

Le compilateur génère les méthodes `equals`,
`hashCode` et `toString`.

Un record peut implémenter une interface, mais ne peut pas hériter d'une classe. Pas de getter, on accède à un champ par une méthode portant le nom du champ:

```
var john = new Person("John", 35);  
john.age(); // 35
```



PATTERN MATCHING INSTANCEOF

évite de devoir caster un objet checké avec
`instanceof`

```
if (animal instanceof Cat) {  
    ((Cat)animal).meow();  
} else if (animal instanceof Dog) {  
    ((Dog)animal).woof();  
}
```

```
if (animal instanceof Cat cat) {  
    cat.meow();  
} else if (animal instanceof Dog dog) {  
    dog.woof();  
}
```

JAVA 17 - SEP 2021

Version "LTS" supportée par Oracle jusqu'en
septembre 2026

SEALED CLASSES

Permet de contrôler l'arbre d'héritage.

```
public abstract sealed class Person
    permits Employee, Manager {
    //...
}

public final class Employee extends Person {}

public final class Manager extends Person {}

// erreur de compilation
public final class OtherPerson extends Person {}
```

JAVA 21 - SEP 2023

SEQUENCED COLLECTIONS

Représente une séquence ordonnée d'éléments.

```
interface SequencedCollection<E> extends Collection<E> {  
    SequencedCollection<E> reversed();  
    void addFirst(E);  
    void addLast(E);  
    E getFirst();  
    E getLast();  
    E removeFirst();  
    E removeLast();  
}
```

List<E>, LinkedList<E>, ArrayList<E>,
SortedSet<E>, LinkedHashSet<E>,
SortedMap<E>... implémentent

PATTERN MATCH FOR SWITCH

```
static String formatterPatternSwitch(Object obj) {  
    return switch (obj) { // évite de faire des instances of  
        case Integer i -> String.format("int %d", i);  
        case Long l     -> String.format("long %d", l);  
        case Double d    -> String.format("double %f", d);  
        case String s    -> String.format("String %s", s);  
        default          -> obj.toString();  
    };  
}
```

PATTERN MATCH FOR SWITCH

with case refinement

```
static void testStringEnhanced(String response) {  
    switch (response) {  
        case null -> { }  
        case "y", "Y" -> { // des constantes, comme d'habitude  
            System.out.println("You got it");  
        }  
        case "n", "N" -> {  
            System.out.println("Shame");  
        }  
        case String s // pattern matching sur le type  
        when s.equalsIgnoreCase("YES") -> { // refinement `when`  
            System.out.println("You got it");  
        }  
        case String s  
        when s.equalsIgnoreCase("NO") -> {
```

VIRTUAL THREADS (GREEN THREADS)

Les modèles de programmation concurrente sont souvent **thread-per-request**. Un **thread OS** coûte 1MB de RAM (par défaut), et un context-switch (Kernel).

```
$ java -XX:+PrintFlagsFinal --version | grep -i ThreadStack
    intx CompilerThreadStackSize           = 1024
    intx ThreadStackSize                   = 1024
    intx VMThreadStackSize                  = 1024
```

Combien de Threads on peut instancier sur une machine ?

VIRTUAL THREADS (GREEN THREADS)

```
/**  
 * A thread that is scheduled by the Java virtual machine  
 * rather than the operating system.  
 */  
final class VirtualThread extends Thread {}
```

Créer un Virtual Thread:

```
Thread.ofVirtual().name("Duke").start(runnable);  
Thread.ofVirtual().name("Duke").unstarted(runnable);  
  
Thread.ofPlatform().name("Duke OS").start(runnable);  
Thread.ofPlatform().name("Duke OS").unstarted(runnable);
```

VIRTUAL THREADS (GREEN THREADS)

Avec un `ExecutorService`:

```
var executor = Executors.newVirtualThreadPerTaskExecutor();  
var future = executor.submit(runnable);
```

TP - JAVA !



FIN DU COURS