Comparación entre Java 8, 9 y 17

Introducción

En este documento voy a explicar con mis propias palabras las diferencias más importantes entre Java 8, Java 9 y Java 17. La idea no es hacer una lista de cosas técnicas copiadas de internet, sino entender qué cambió, para qué sirve y cómo podría aplicarse en un proyecto real.

Java 8 (2014)

Java 8 es una de las versiones más importantes en la historia del lenguaje. Fue un antes y un después porque introdujo las expresiones lambda y los Streams. Esto significa que se puede trabajar con colecciones de datos de una forma más simple y clara.

Ejemplo con Java 8:

```
List<String> nombres = Arrays.asList("Ana", "Luis", "Pedro");
nombres.forEach(nombre -> System.out.println(nombre));
```

En lugar de usar un ciclo for traditional, ahora podemos recorrer una lista con `forEach` y una lambda. Esto hizo que el código se viera más moderno y parecido a otros lenguajes.

Java 9 (2017)

Java 9 no fue tan revolucionario como Java 8, pero sí agregó cambios muy importantes para proyectos grandes. El principal fue el sistema de módulos (Project Jigsaw), que permite organizar mejor un programa en partes independientes. Esto es útil cuando un sistema crece demasiado y se necesita tener orden.

Ejemplo con Java 9:

```
module com.miapp {
    requires java.sql;
    exports com.miapp.modelo;
}
```

Con este tipo de archivo `module-info.java` podemos definir qué partes de nuestra aplicación se usan afuera y cuáles son internas.

Java 17 (2021)

Java 17 es una versión LTS (soporte a largo plazo), lo que significa que muchas empresas lo adoptan porque tendrá actualizaciones y soporte por más tiempo. Una de las novedades que más me llamó la atención fueron los Records, que sirven para definir clases de solo datos de forma súper corta.

Ejemplo con Java 17:

```
public record Persona(String nombre, int edad) {
}

public class Main {

    public static void main(String[] args) {
        Persona p = new Persona("Ana", 25);
        System.out.println(p.nombre());
    }
}
```

En lugar de escribir una clase completa con constructores, getters y toString, ahora con una sola línea ya tenemos todo listo.

Conclusión personal

Después de revisar las tres versiones, veo que cada una tuvo su papel. Java 8 modernizó la forma de programar, Java 9 ayudó a mantener proyectos grandes en orden y Java 17 consolidó el lenguaje con mejoras que hacen el código más simple y con buen rendimiento. En mi opinión, para un proyecto nuevo conviene usar Java 17 porque es estable y tiene soporte LTS, pero también es importante entender Java 8 porque muchas empresas todavía lo usan.