

De Java 8 a Java 18. Lo que debes de saber.

Isaac Ruiz Guerra
(**RuGI**)
<https://twitter.com/rugi>



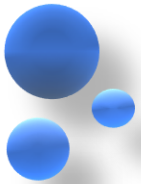
SPS

<https://www.spsolutions.com.mx/>



Agenda

- Preámbulo.
- Cambios importantes en el lenguaje.
- Herramientas del JDK.
- La importancia de moverse hacia una versión más reciente.
- No perder de vista.



Aplausos. Imagen: Viralplus

Preámbulo.

Cadencia de versiones

Oracle sacando versiones de java cada 6 meses:



A considerar:

- Desde java 9, Oracle modificó la cadencia de versiones de java a 6 meses.
- Cada 6 meses tendremos una nueva versión.
- Habrá versiones que tendrán soporte por más tiempo (LTS)

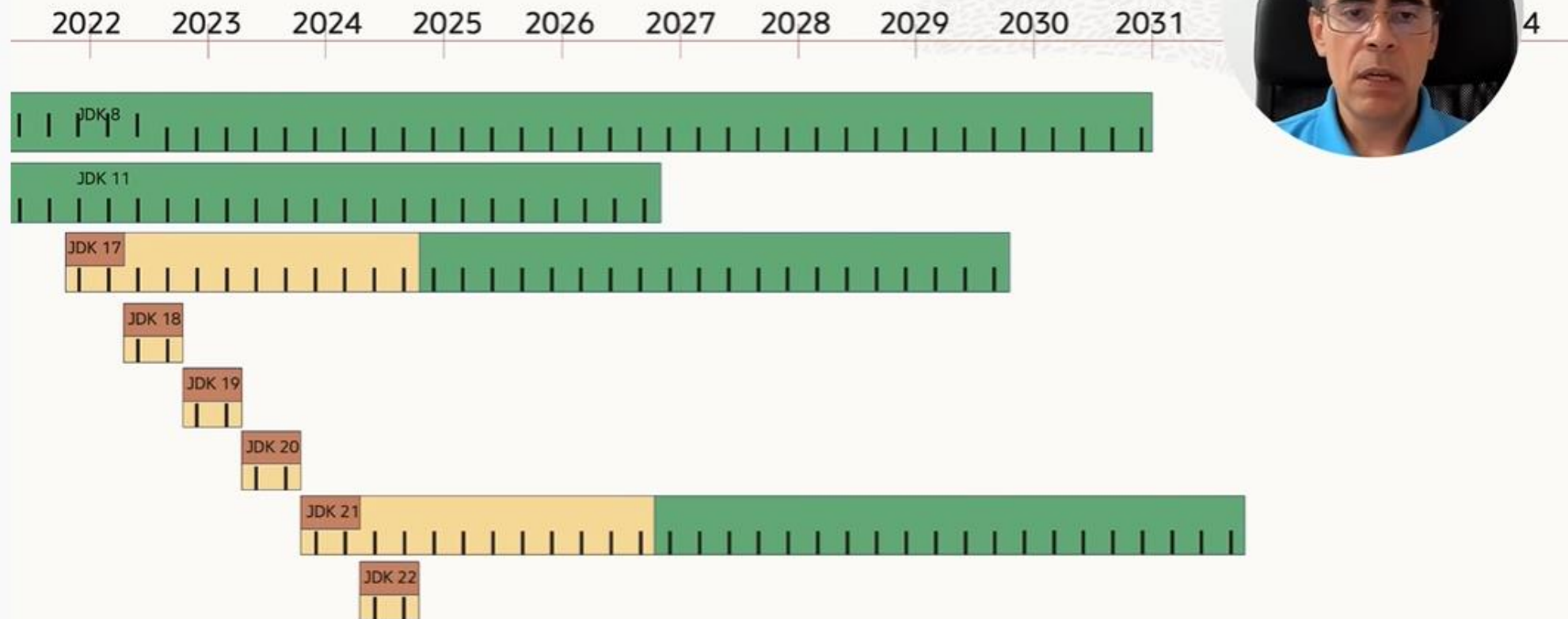
<https://twitter.com/mapacherants/status/1569694309812772864>

Es necesario tener presente las fechas y cuales son LTS (2 años)

Released			
Date	Type	Version	Other Information
2022-09-20	Feature	19	Documentation, JSR 394, Configurations
2022-08-18	Patch	18.0.2	
"	"	17.0.4.1	
"	"	11.0.16.1	
2022-07-19	CPU	18.0.2	Risk Matrix
"	"	17.0.4	
"	"	11.0.16	
"	"	8u341	
"	"	7u351	
2022-05-02	Patch	18.0.1.1	
"	"	17.0.3.1	
"	"	11.0.15.1	
"	"	8u333	
"	"	7u343	
2022-04-19	CPU	18.0.1	Risk Matrix
"	"	17.0.3	
"	"	11.0.15	
"	"	8u331	
"	"	7u341	
2022-03-22	Feature	18	Documentation, JSR 393, Configurations
2022-01-18	CPU	17.0.2	Risk Matrix
"	"	11.0.14	
"	"	8u321	
"	"	7u331	
2021-10-19	CPU	17.0.1	Risk Matrix
"	"	11.0.13	
"	"	8u311	
"	"	7u321	
2021-09-14	Feature	17 LTS	Documentation, JSR 392, Configurations

Planned ⁽¹⁾			
Date	Type	Version	Other Information
2024-09-17	Feature	23	
2024-07-16	CPU		22.0.2, 21.0.4, 17.0.12, 11.0.24, 8u421
2024-04-16	CPU		22.0.1, 21.0.3, 17.0.11, 11.0.23, 8u411
2024-03-19	Feature	22	
2024-01-16	CPU		21.0.2, 17.0.10, 11.0.22, 8u401
2023-10-17	CPU		21.0.1, 17.0.9, 11.0.21, 8u391
2023-09-19	Feature	21 LTS	
2023-07-18	CPU		20.0.2, 17.0.8, 11.0.20, 8u381
2023-04-18	CPU		20.0.1, 17.0.7, 11.0.19, 8u371
2023-03-21	Feature	20	Early Access
2023-01-17	CPU		19.0.2, 17.0.6, 11.0.18, 8u361
2022-10-18	CPU		19.0.1, 17.0.5, 11.0.17, 8u351

Java Release Model and Licenses - Next



All updates are included without a Java SE Subscription, when running on Oracle Cloud.

Copyright © 2022, Oracle and/or its affiliates

July 2022



JLS, JSR, JCP, JEP.

- JLS.
 - Java Language Specification.

EL JLS es el documento que describe al lenguaje en sí mismo, básicamente este documento describe qué es válido o no al momento de escribir código.

La definición de la **sintaxis** y **semántica** de lo que escribimos en el código. *¿Es válido esa sentencia? ¿Eso es un número entero? ¿así se declara un arreglo?* Todo eso viene ahí.

JLS, JSR, JCP, JEP.

- JSR.
 - Java Specification Request.

Todo en java tiene un JSR, los JSR son documentos formales y estructurados de varias páginas que se utilizan para definir una nueva característica o una tecnología a la plataforma java. ¿Cómo funciona la mensajería en java?

Se especifica en el: JSR 368: Java Message Service 2.1. ¿Cómo procesar JSON con el lenguaje? Se especifica en: JSR 374: Java API for JSON Processing 1.1., cualquier implementación que pienses tiene un JSR. Una vez que se tiene el JSR, se construyen distintas implementaciones que podemos utilizar. El mismo lenguaje tiene su propio JSR. JSR 392: Java SE 17.

JLS, JSR, JCP, JEP.

- JCP.
 - Java Community Process.

El JCP es el proceso encargado de regular la evolución de la plataforma java, a través de un grupo de expertos, se revisa cada JSR y se determina cuáles son los que se incorporan a la plataforma.

El grupo de expertos lo conforman: empresas, instituciones educativas y personas. El JCP tiene como insumo principal de trabajo a los JSRs. El proceso tiene un ciclo de vida definido para la aprobación de cada JSR (sí, el mismo JCP tiene su propio JSR: JSR 215: Java Community Process version 2.6).

JLS, JSR, JCP, JEP.

- JEP.
 - JDK Enhancement Proposal

Los JEPs son propuestas de cambios tanto en el lenguaje como en la plataforma, son un tanto más ligeros e informales en su presentación y tienen como propósito la exploración de dichas propuestas (extender la sintaxis del switch o proponer un cambio al GC son ejemplos de JEP).

Un EJP una vez maduro debe agregarse a un JSR y así pasar por el JCP para su incorporación a la plataforma. Al igual que los JSRs tienen asociados un número.

JLS, JSR, JCP, JEP.

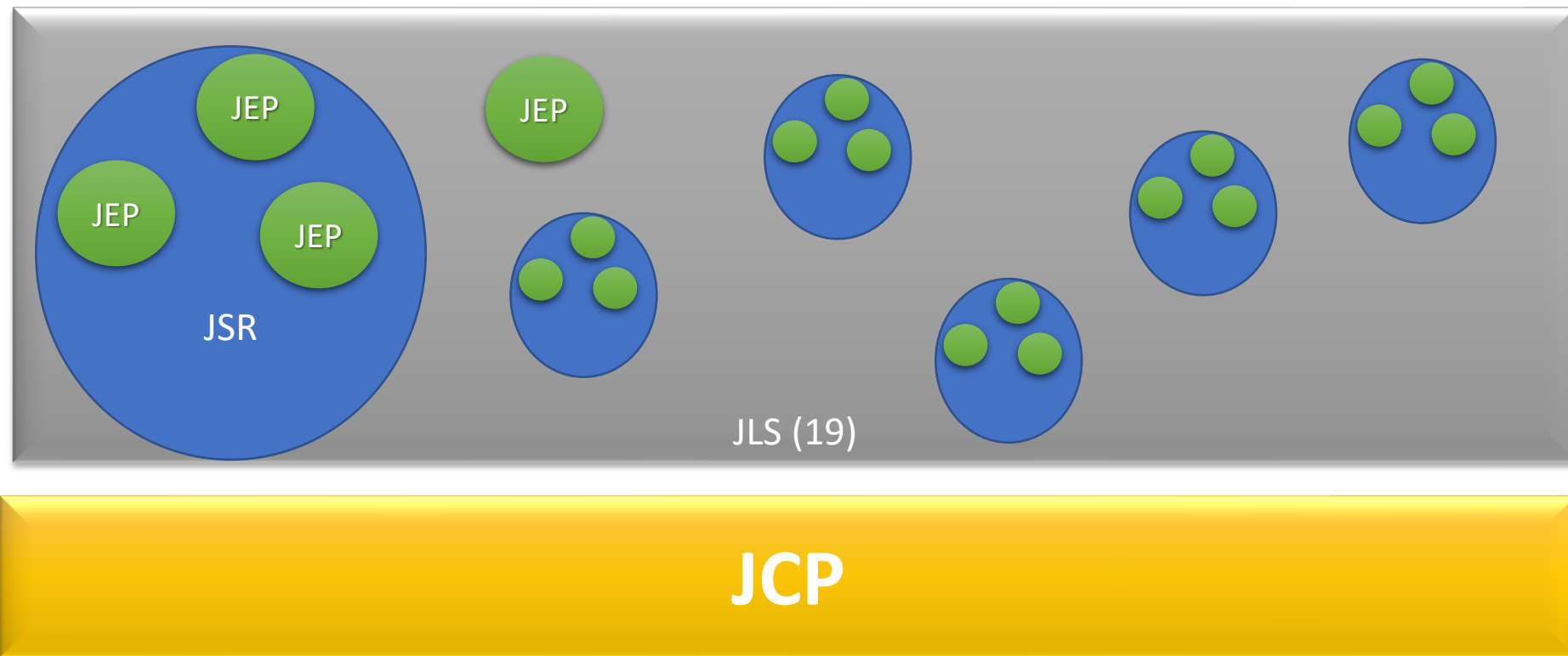
- JEP.
 - JDK Enhancement Proposal

Existen 3 JEP fundamentales: **JEP 1**: JDK Enhancement-Proposal & Roadmap Process, define el proceso *per se* del JEP, el **JEP 2**: JEP Template el template que se usa para un JEP y el **JEP 0**: JEP Index muestra el índice de todos los EJPs.

La mayoría de los JEPs aceptados se agregan al JSR del JLS en su versión próxima inmediata, por ejemplo El **JEP 409**: Sealed Classes y el **JEP 356**: Enhanced Pseudo-Random Number Generators fueron agregado a la versión 17 de java (JSR 392: Java SE 17).

<https://openjdk.org/jeps/0>

JLS, JSR, JCP, JEP.



<https://openjdk.org/jeps/0>

JDK 19 Release Notes

These notes describe important changes, enhancements, removed APIs and features, deprecated APIs and features, and other information about JDK 19 and Java SE 19. In some cases, the descriptions provide links to additional detailed information about an issue or a change. This page does not duplicate the descriptions provided by the [Java SE 19 \(JSR 394\) Platform Specification](#), which provides informative background for all specification changes and might also include the identification of removed or deprecated APIs and features not described here. The Java SE 19 (JSR 394) specification provides links to:

- **Annex 1:** The complete [Java SE 19 API Specification](#).
- **Annex 2:** An [annotated API specification](#) showing the exact differences between Java SE 17 and Java SE 19. Informative background for these changes may be found in the list of approved Change Specification Requests for this release.
- **Annex 3:** Java SE 19 Editions of [The Java Language Specification](#) and [The Java Virtual Machine Specification](#). The Java SE 19 Editions contain all corrections and clarifications made since the [Java SE 17 Editions](#), as well as additions for new features.

You should be aware of the content in the [Java SE 19 \(JSR 394\) specification](#) as well as the items described in this page.

The descriptions on this Release Notes page also identify potential compatibility issues that you might encounter when migrating to JDK 19. The [Kinds of Compatibility](#) page on the OpenJDK wiki identifies the following three types of potential compatibility issues for Java programs that might be used in these release notes:

- **Source:** Source compatibility preserves the ability to compile existing source code without error.
- **Binary:** Binary compatibility is defined in [The Java Language Specification](#) as preserving the ability to link existing class files without error.

See [CSRs Approved for JDK 19](#) for the list of CSRs closed in JDK 19 and the [Compatibility & Specification Review \(CSR\)](#) page on the OpenJDK wiki for general information about compatibility.

The full version string for this release is build 19+36 (where "+" means "build"). The version number is 19.

IANA Data 2022a

JDK 19 contains IANA time zone data versions 2022a. For more information, refer to [Timezone Data Versions in the JRE Software](#).

The following are highlights of significant changes

JEP 405 Record Patterns (Preview)

Enhance the Java programming language with *record patterns* to deconstruct record values. Record patterns and type patterns can be nested to enable a powerful, declarative, and composable form of data navigation and processing. This is a [preview language feature](#).

JEP 424 Foreign Function & Memory API (Preview)

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI. This is a [preview API](#).

JEP 425 Virtual Threads (Preview)

Introduce *virtual threads* to the Java Platform. Virtual threads are lightweight threads that dramatically reduce the effort of writing, maintaining, and observing high-throughput concurrent applications. This is a [preview API](#).

JEP 426 Vector API (Fourth Incubator)

Introduce an API to express vector computations that reliably compile at runtime to optimal vector instructions on supported CPU architectures, thus achieving performance superior to equivalent scalar computations.

JEP 427 Pattern Matching for switch (Third Preview)

Cambios en el lenguaje.

Interface default, Static Methods y Interface private Methods.

Esto:

```
public interface Downloadable {  
    boolean isTxt();  
    boolean isBinary();  
    String version();  
}
```

Realmente es:

```
public interface Downloadable {  
    public abstract boolean isTxt();  
    public abstract boolean isBinary();  
    public abstract String version();  
}
```

Interface default, Static Methods y Interface private Methods.

Default Method:

```
public interface Downloadable {  
    boolean isTxt();  
    boolean isBinary();  
    default String version() {  
        return "v1.0";  
    }  
}
```

Static Method:

```
public interface Downloadable {  
    boolean isTxt();  
    boolean isBinary();  
    default String version() {  
        return Downloadable.message();  
    }  
    static String message() {  
        return "v1.0";  
    }  
}
```

Con los métodos de tipo *static*, podemos tener implementaciones que no pertenezcan a ningún objeto (no podemos decir: implementaciones *a nivel de clase* ya que se trata de una **interface**).

Interface default, Static Methods y Interface private Methods.

Estos cambios generaron críticas ya que esto aleja un poco a java de la POO pues, le estamos dando comportamiento a una interface cuando en esencia esa es labor de las clases.

Java 9. Privated Method.

```
public interface Downloadable {  
    boolean isTxt();  
    boolean isBinary();  
    default String version() {  
        return Downloadable.message();  
    }  
    private static String message() {  
        return "v1.0";  
    }  
}
```

Inferencia de tipos.

El tipo de dato y el valor a asignar deben ser compatibles.

TipoDeLaVariable *NombreDeLaVariable* = **ValorAasignar**;

Ahora

String nombre = "Juan"

→

var nombre = "Juan"

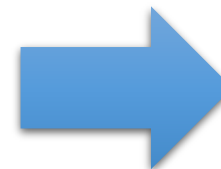
Inferencia de tipos.

```
import java.util.ArrayList;

public class VarTest {

    public static void main(String[] args) {
        // int
        var x = 100;
        // double
        var y = 1.90;
        // char
        var z = 'a';
        // String
        var p = "Hola";

        // boolean var q = false;
        //Objetos var lista = new ArrayList<String>();
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
        System.out.println(p);
        System.out.println(q);
        System.out.println(lista);
    }
}
```



```
100
1.9
a
Hola
false
[]
```

Inferencia de tipos.

Entonces
¿Perdemos lo fuertemente tipado?

Inferencia de tipos.

1. La inferencia de tipos no se permite a nivel de clase

```
import java.util.ArrayList;

public class VarTest {

    var nombre = "Pedro"; //Esto no se permite. Debe ser local. NO a nivel de clase.

    public static void main(String[] args) {
        // int var x = 100; // double
        .....
    }

}
```

Inferencia de tipos.

2. Una vez inferido el tipo, no se puede cambiar.

```
var k = 2;  
k= 1.3; //Esto no es posible, k fue definido como tipo int
```

3. La inferencia se realiza con la inicialización, no se puede realizar si ésta no existe:

```
var x;  
x=100; //Esto no es posible, se requiere declarar e iniciar en la misma  
línea
```

4. No se puede usar para recibir el resultado de una expresión Lambda:

```
var obj = (a, b) -> (a + b); //Una expresión lambda necesita  
explícitamente un tipo objetivo.
```


Inferencia de tipos.

5. No se permite usar `var` para definir un parámetro de entrada:

```
int suma (var i, int j){ //No se permite aquí.  
    return i+j;  
}
```

6. Tampoco se permite usar como valor de retorno en la declaración de un método:

```
var suma (int i, int j){ // No se permite aquí.  
    return i+j;  
}
```

Text Blocks.

JEP 378

Antes:

```
String mensaje = "<persona>\n" +  
    "  <nombre>Juan</nombre>\n" +  
    "  <apellido>Torres</apellido>\n" +  
    "</persona>";  
System.out.println(mensaje);
```

Genera

```
<persona>  
  <nombre>Juan</nombre>  
  <apellido>Torres</apellido>  
</persona>
```

JEP 355

Text Blocks.

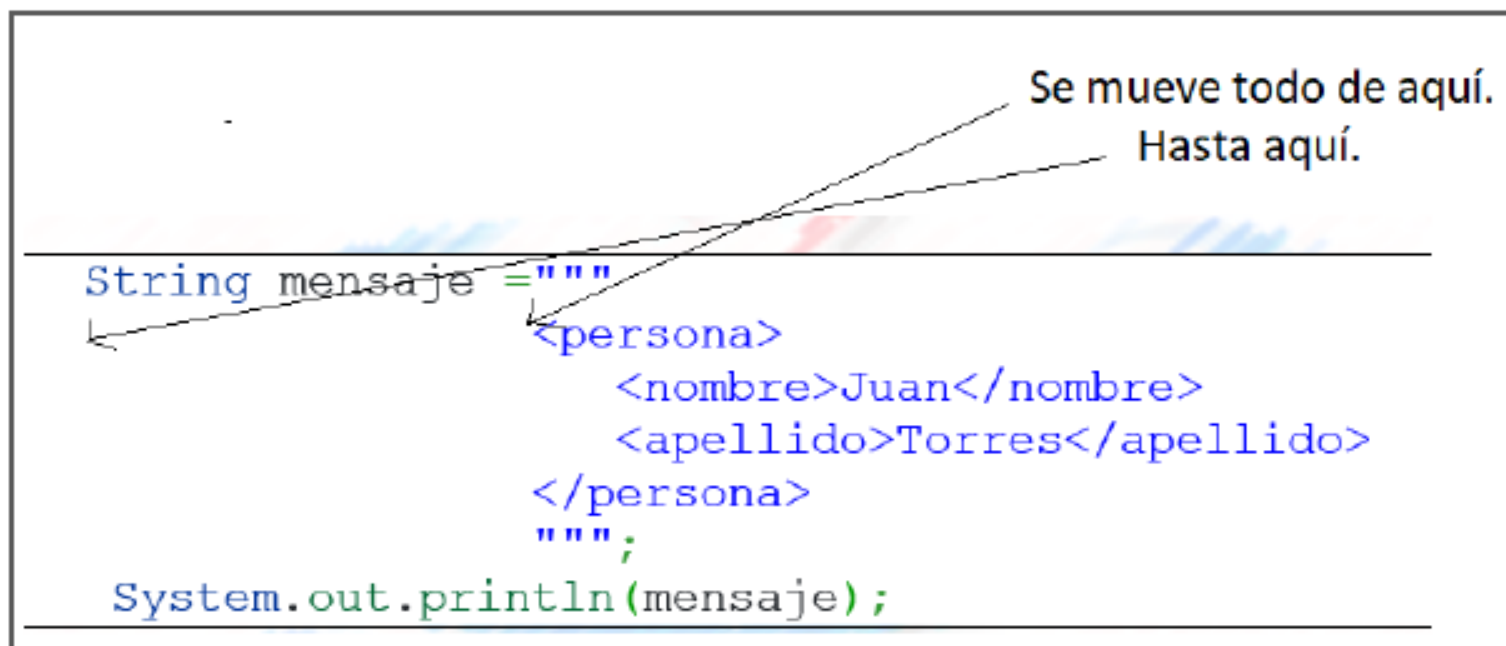
Ahora:

```
String mensaje = """
    <persona>
      <nombre>Juan</nombre>
      <apellido>Torres</apellido>
    </persona>
    """;
System.out.println(mensaje);
```



```
<persona>
  <nombre>Juan</nombre>
  <apellido>Torres</apellido>
</persona>
```

Text Blocks.



The diagram illustrates the syntax of a text block in Java. It shows a code snippet within a rectangular frame. The code is: `String mensaje = "<persona>\n <nombre>Juan</nombre>\n <apellido>Torres</apellido>\n</persona>\n";` followed by `System.out.println(mensaje);`. Annotations include: a horizontal line above the opening quote of the text block; a horizontal line below the closing quote; an arrow pointing from the text "Se mueve todo de aquí." to the opening quote; and another arrow pointing from the text "Hasta aquí." to the closing quote. The text "Se mueve todo de aquí." is positioned above the first line, and "Hasta aquí." is positioned above the second line.

```
String mensaje = "  
    <persona>  
        <nombre>Juan</nombre>  
        <apellido>Torres</apellido>  
    </persona>  
    ";  
System.out.println(mensaje);
```

Se mueve todo de aquí.
Hasta aquí.

Text Blocks.

Veamos este *text block*, si ejecutamos:

```
String frase = """
    Este es un texto con "comillas" dobles
    Comillas 'sencillas'
    Y ahora 3 comillas \"""
    """;
System.out.println(frase);
```

Obtendremos:

```
Este es un texto con "comillas" dobles
Comillas 'sencillas'
Y ahora 3 comillas """
```

Switch Expressions

```
public static String diaLetras1(int diaNumero) {  
    String dia = null;  
    switch (diaNumero) {  
        case 1:  
            dia = "Lunes"; break;  
        case 2:  
            dia = "Martes"; break;  
        case 3:  
            dia = "Miercoles"; break;  
        case 4:  
            dia = "Jueves"; break;  
        case 5:  
            dia = "Viernes"; break;  
        case 6:  
            dia = "Sábado"; break;  
        case 7:  
            dia = "Domingo"; break;  
        default:  
            String mensaje = "Mensaje de error:";  
            String mensaje2 = recuperarMensajeCompleto();  
            dia = mensaje + mensaje2; break;  
    }  
    return dia;  
}
```



```
public static String diaLetras2(int diaNumero) {  
    String dia = switch (diaNumero) {  
        case 1 -> "Lunes";  
        case 2 -> "Martes";  
        case 3 -> "Miercoles";  
        case 4 -> "Jueves";  
        case 5 -> "Viernes";  
        case 6 -> "Sábado";  
        case 7 -> "Domingo";  
        default -> {  
            String mensaje = "Mensaje de error:";  
            String mensaje2 =  
                recuperarMensajeCompleto();  
            yield mensaje + mensaje2;  
        }  
    };  
    return dia;  
}
```

JEP 325
JEP 354
JEP 361

Records

```
public class Cliente {  
    private String nombre;  
    private int edad;  
    private boolean preferente;  
    public Cliente(String nombre, int edad, boolean preferente)  
    {  
        super();  
        this.nombre = nombre;  
        this.edad = edad;  
        this.preferente = preferente;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getEdad() {  
        return edad;  
    }  
  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
  
    public boolean isPreferente() {  
        return preferente;  
    }  
  
    public void setPreferente(boolean preferente) {  
        this.preferente = preferente;  
    }  
}
```

Estas clases son conocidas como Plain Old Java Objects (POJO) o Data Transfer Object (DTO) ya que usualmente se usan para pasar información de una capa de abstracción hacia otra.

Hablando de
Setters
Y
Getters

JEP 384 JEP 395 JEP 359

Records

```
public class Cliente {
    private String nombre;
    private int edad;
    private boolean preferente;
    public Cliente(String nombre, int edad, boolean preferente)
    {
        super();
        this.nombre = nombre;
        this.edad = edad;
        this.preferente = preferente;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public boolean isPreferente() {
        return preferente;
    }

    public void setPreferente(boolean preferente) {
        this.preferente = preferente;
    }
}
```

```
@Override public String toString() {
    return new StringBuilder().append(this.nombre).append(":").
        append(this.edad).append(":").
        append(this.preferente).toString();
}

@Override public boolean equals(Object obj) {
    if (obj == this) {
        return true;
    }
    if (!(obj instanceof Cliente)) {
        return false;
    }
    Cliente cliente = (Cliente) obj;
    return cliente.nombre.equals(this.nombre) &&
        cliente.edad == this.edad &&
        cliente.preferente == this.preferente;
}

@Override public int hashCode() {
    return Objects.hash(nombre, "" + edad, "" + preferente);
}
```

¿Por qué es importante implementar esos 3 métodos? Pregúntenle a las implementaciones de Collections y a los métodos: **equals** y en consecuencia a **contains**.

Records

```
public record Cliente(String nombre, int edad, boolean preferente) {  
  
}
```

```
public static void main(String[] args) {  
    Cliente c1 = new Cliente("Juan", 12, true);  
    Cliente c2 = new Cliente("Juan", 12, true);  
    HashSet<Cliente> clientesEnHash = new HashSet<>();  
    List<Cliente> clientesEnLista = new ArrayList<>();  
    clientesEnHash.add(c1);  
    clientesEnLista.add(c1);  
    System.out.println("Cliente.....:" + c1);  
    System.out.println("Son iguales c1 to c2..." + c1.equals(c2));  
    System.out.println("Son iguales c2 to c1..." + c2.equals(c1));  
    System.out.println("Existe cliente en List:" + clientesEnLista.contains(c2));  
    System.out.println("Existe cliente en Hash:" + clientesEnHash.contains(c2));  
    System.out.println("Nombre del cliente....." + c1.nombre());  
    System.out.println("Edad del cliente....." + c1.edad());  
    System.out.println("Es cliente preferente..." + c1.preferente());  
}
```

Records



Cuando ya usas Records.

Pattern Matching para el operador instanceof.

```
# Código en Elixir
iex> {a, b, c} = {:hello, "world", 42}
{:hello, "world", 42}
iex> a
:hello
iex> b
"world"
iex> c
42
```



{a, b, c} = {:hello, "world", 42}

Operador

Hablando de
Tuplas

Pattern Matching para el operador instanceof.

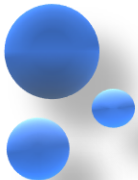
JEP 305
JEP 375
JEP 394

```
if (obj instanceof String) {  
    String s = (String) obj;  
    System.out.println("Tamaño:" + s.length());  
}
```

```
if (obj instanceof String s) {  
    System.out.println("Tamaño:" + s.length());  
}
```


`obj instanceof String s`

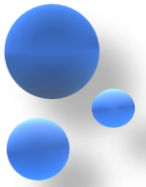
```
if (obj instanceof String s && s.length()>100) {  
    System.out.println("Tamaño muy largo:" + s.length());  
}
```



Herramientas en el JDK.

Herramientas CLI.

- Javap
 - Desensamblar una clase, ya existe desde java 6.
- Jshell
 - Consola RPEL, disponible desde la versión 9
- JLink
 - Crear distintos *runtimes*.
- Jpackage
 - Empaqueta y crea aplicaciones autocontenidas.
 - Linux
 - DEB, RPM
 - Windows
 - MSI, EXE
 - MacOS
 - DMG, PKG
- Flight Recorder 
 - Diagnostico y profiling de aplicaciones java.

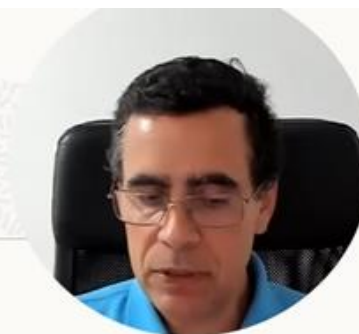
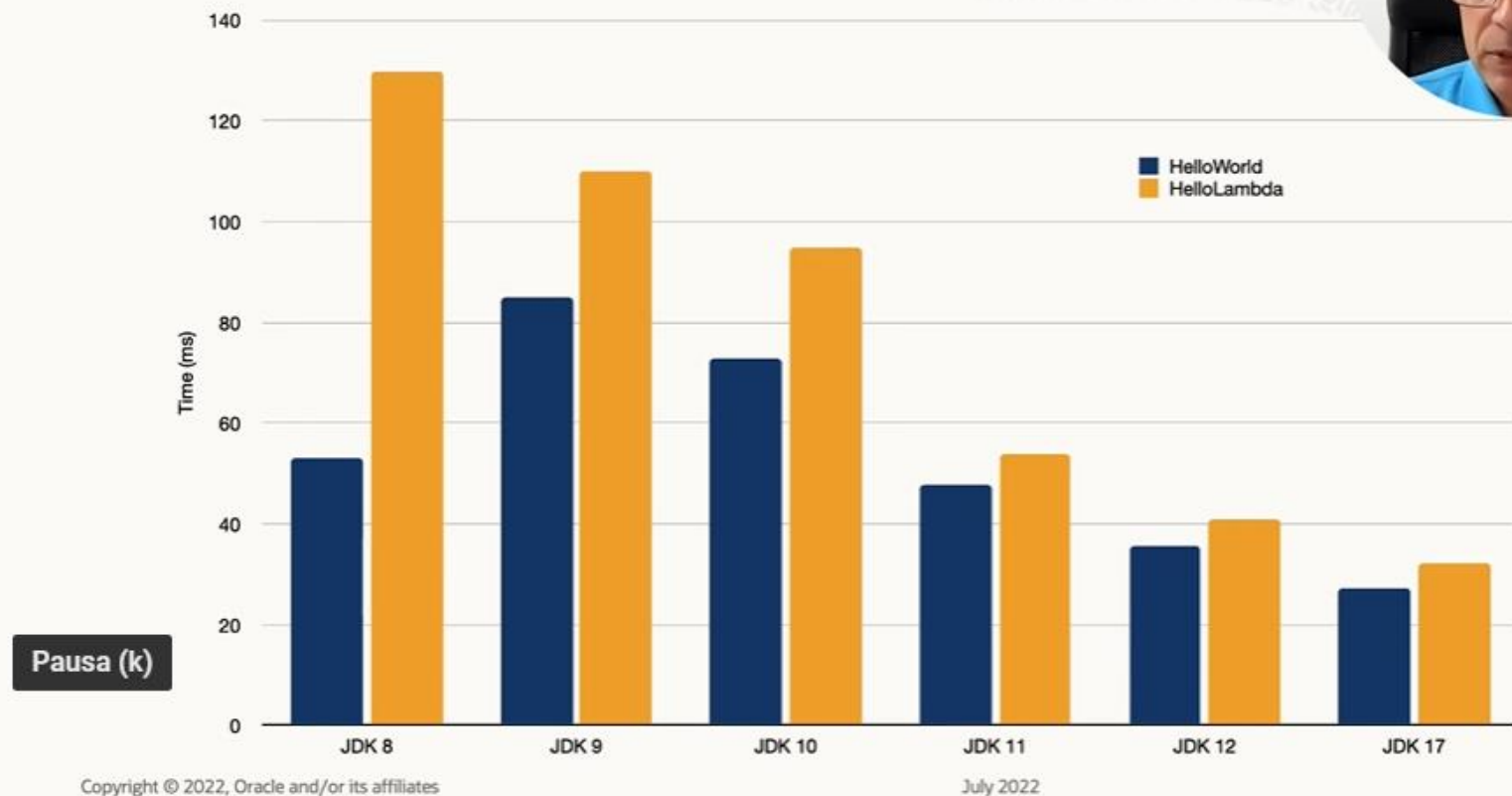


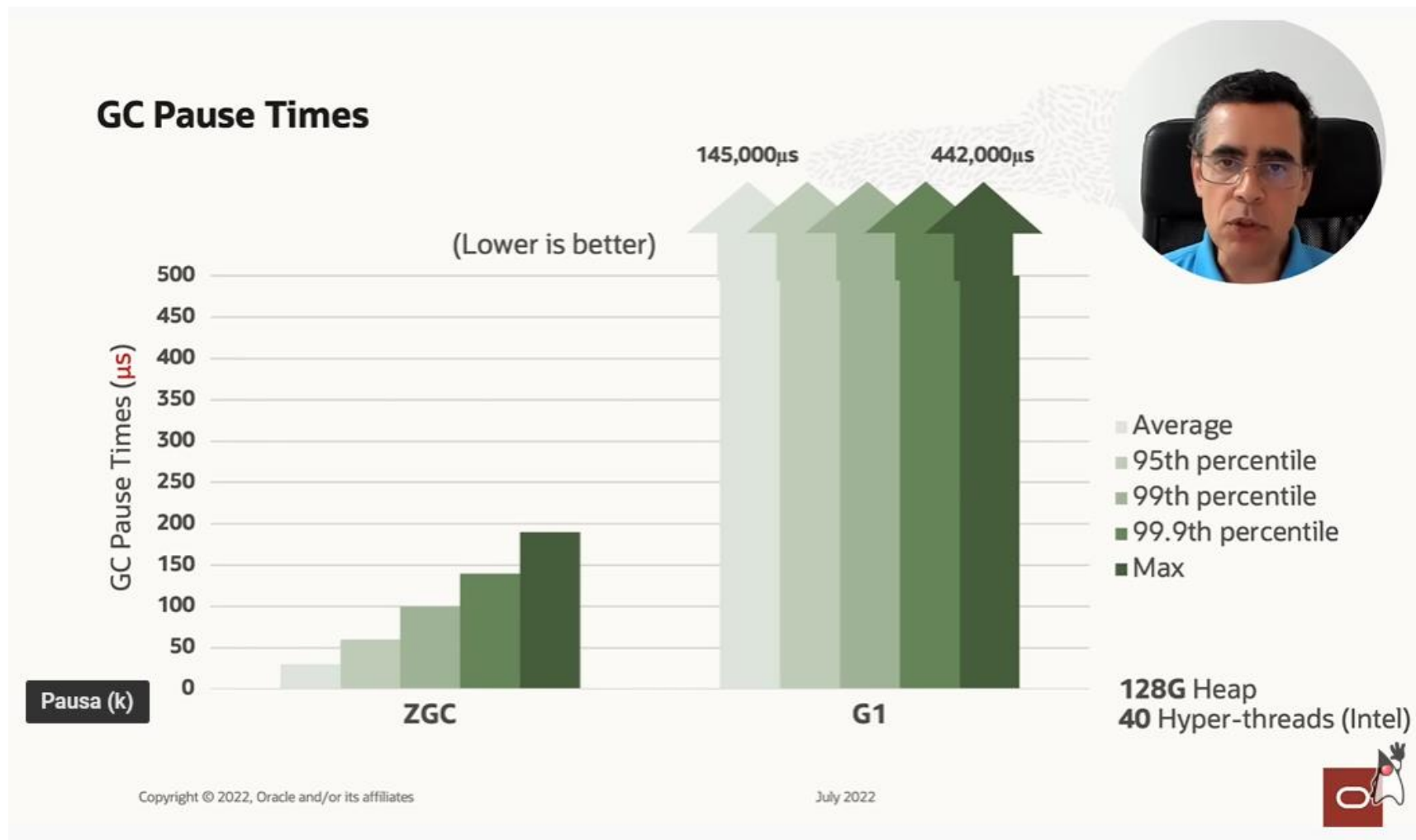
La importancia de moverse a una version Más reciente.

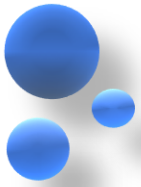
Muévanse sin cambiar código.

- El solo hecho de usar una JVM más reciente ya les ayuda a mejorar.
 - Performance.
 - Garbage Collector.

Startup Time: HelloLambda







No perder de vista.

Las siguientes charlas:

- 3:30 pm Migrando versiones de JDK.
- 4:40 pm Taking off with JDK Flight Recorder.
- 5:30 pm Cierre.
- 6:30 pm Fiesta Post Evento.

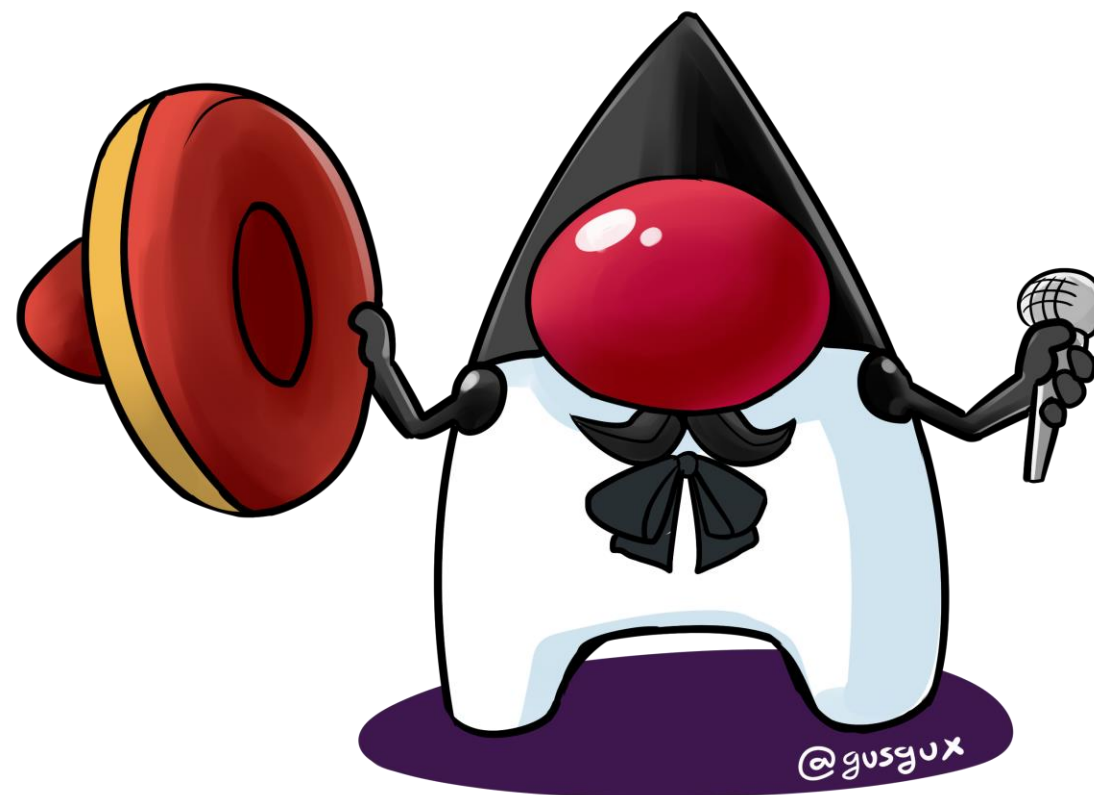


<https://jdk.java.net/19/>

Quédense hasta el final.



<https://optimainfinito.com/2014/02/10-diferencias-entre-equipos-y-redes-productivas.html>



iii Gracias !!!