

NOVEDADES EN JAVA 9

¡BIENVENIDOS!

Soy Luis Miguel López Magaña

Me dedico a ser profesor en ciclos formativos



www.linkedin.com/in/luismi-lopez

AGENDA

1. Cambio de filosofía en Java
2. Modularización
3. JShell, un REPL para Java
4. Nuevas funcionalidades
5. Otras novedades

JAVA CAMBIA DE FILOSOFÍA

Java hasta la versión 8

- Java se origina en una empresa llamada SUN MICROSYSTEMS
- La versión JDK 1.0 se lanza en el año 1995-96
- Primeras versiones importantes:
 - J2SE 1.2: Dic-1998.
 - J2SE 1.4: Feb-2002
- Liberación de versiones cada ¿2-3 años?

Java hasta la versión 8

- Atasco entre las versiones 6 y 7:
 - Java SE 6: Liberada en Dic-2006
 - Java SE 7: Liberada en ¡Jul-2011!
- ¿Algún motivo?
 - Oracle adquiere SUN en 2010

Java hasta la versión 8

- Java 8
 - Se lanza en Mar-2014
 - Incluye algunas mejoras que se iban a incluir en Java 7
 - Primera gran versión liberada por Oracle
 - Cambios significativos: lambdas, fechas, streams, ...

Soporte de versiones hasta Java 8

| Versión | Fecha lanzamiento | Fecha fin soporte | Fecha fin soporte extendido |
|------------------------|-------------------|---------------------------------------|-----------------------------|
| Java SE 6 | Diciembre de 2006 | Abril de 2013 | Diciembre de 2018 |
| Java SE 7 | Julio de 2011 | Abril de 2015 | Julio de 2022 |
| Java SE 8 (LTS) | Marzo de 2014 | Enero de 2019* Diciembre de 2020** | Diciembre de 2030 |
| ... | ... | ... | ... |

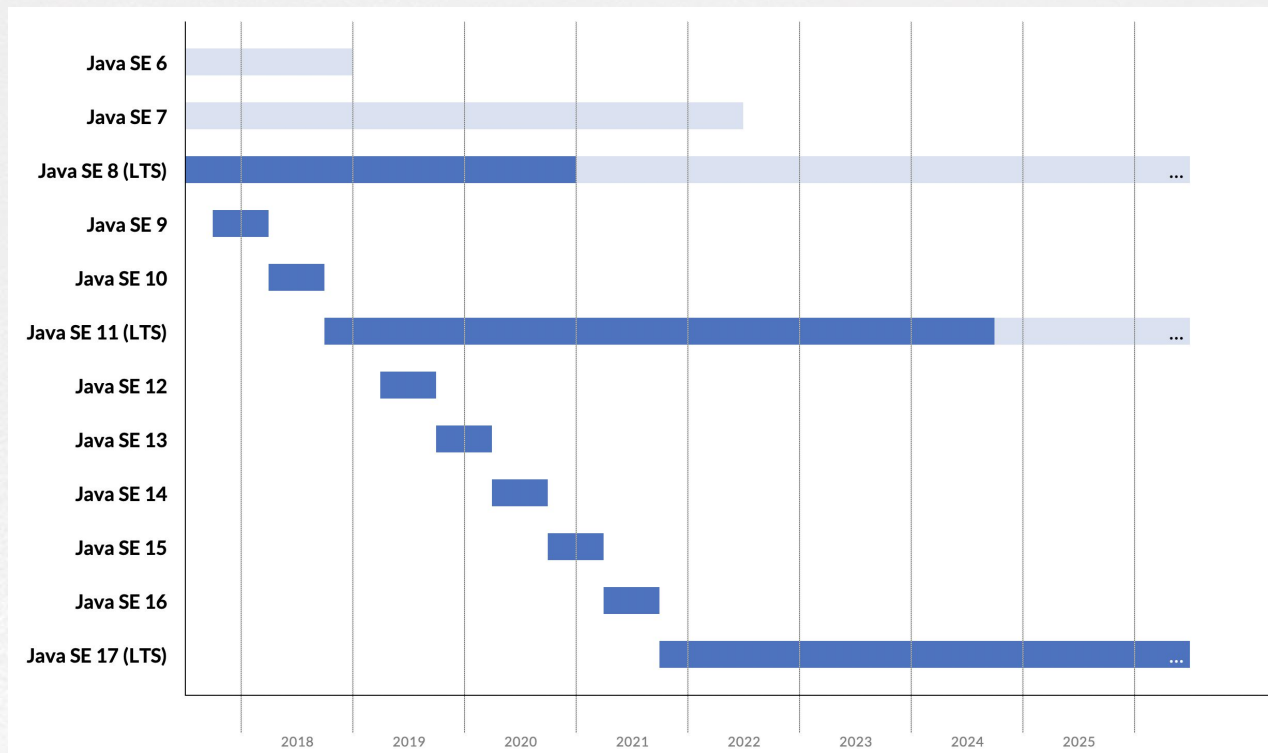
* Para uso comercial

** Para uso personal

Cambios a partir de Java 9: *Release train*

- Java SE 9 se libera en Septiembre de 2017
 - 3 años después de Java SE 8
- A partir de esta versión
 - ~~○ Las nuevas versiones se liberan por funcionalidades~~
 - **Las nuevas versiones se liberan cada 6 meses.**
 - **Solo tendrán soporte durante 6 meses.**
 - **Cada año y medio se libera una versión LTS.**

Cambios a partir de Java 9: *Release train*



Cambios en las *licencias*

- ¿Java es gratis?
 - Java es libre, pero los binarios y las compilaciones no.
- A partir de 2019...
 - Los binarios de Oracle JDK son de pago si se quieren usar en producción.
 - No se emiten actualizaciones públicas para usuarios comerciales de Java 8.

Cambios en las *licencias*

- Consecuencias
 - El software es gratuito
 - En el ámbito empresarial, se paga por el soporte.
- ¿Hay alternativas? Sí. Los binarios de Oracle JDK no son los únicos.
 - OpenJDK de Oracle (licencia GPL)
 - AdoptOpenJDK
 - Zulu
 - ...

Cambios en las *licencias*

- ¿Qué binarios debería usar?
- Ámbito educativo
 - Oracle JDK (sin cargo).
 - Oracle OpenJDK.
 - AdoptOpenJDK
- Ámbito empresarial
 - Oracle JDK: licencia de pago con soporte (extendido).
 - AdoptOpenJDK (varias empresas).
 - Azul Zulu JDK (Amazon).

MODULARIZACIÓN: PROYECTO JIGSAW

MOTIVACIÓN 1: MEJORAR LA ENCAPSULACIÓN

- Java tiene un mecanismo de encapsulación de clases: *public*, *private*, *protected* y *default*.
- Esto, para un solo fichero JAR, puede resultar suficiente.
- Sin embargo, si usamos varios ficheros JAR (i.e. un proyecto con Spring) se vuelve insuficiente.
- ¿Por qué?

MOTIVACIÓN 1: MEJORAR LA ENCAPSULACIÓN

- CLASSPATH: conjunto de rutas donde se va a buscar una clase cuándo hace falta. Se buscan secuencialmente y se para cuando se encuentra una que satisfaga la búsqueda.
 - Problemas con diferentes versiones.
 - Errores en tiempo de ejecución.
- Todas las clases públicas que se incorporan a un proyecto son, básicamente, accesibles desde cualquier punto del proyecto.
- Se puede acceder a clases que, en un principio, no fueron pensadas para ser utilizadas *por cualquiera*.

MOTIVACIÓN 1: MEJORAR LA ENCAPSULACIÓN

- *Un clásico*
 - Definimos una serie de interfaces y sus implementaciones.
 - Queremos ofrecer las interfaces, pero ocultar las implementaciones.
 - Si distribuimos un *.jar*, posiblemente cualquiera pueda usar las implementaciones en lugar de las interfaces.
- Ejemplos: *sun.misc.Unsafe* o *sun.misc.BASE64Encoder*.

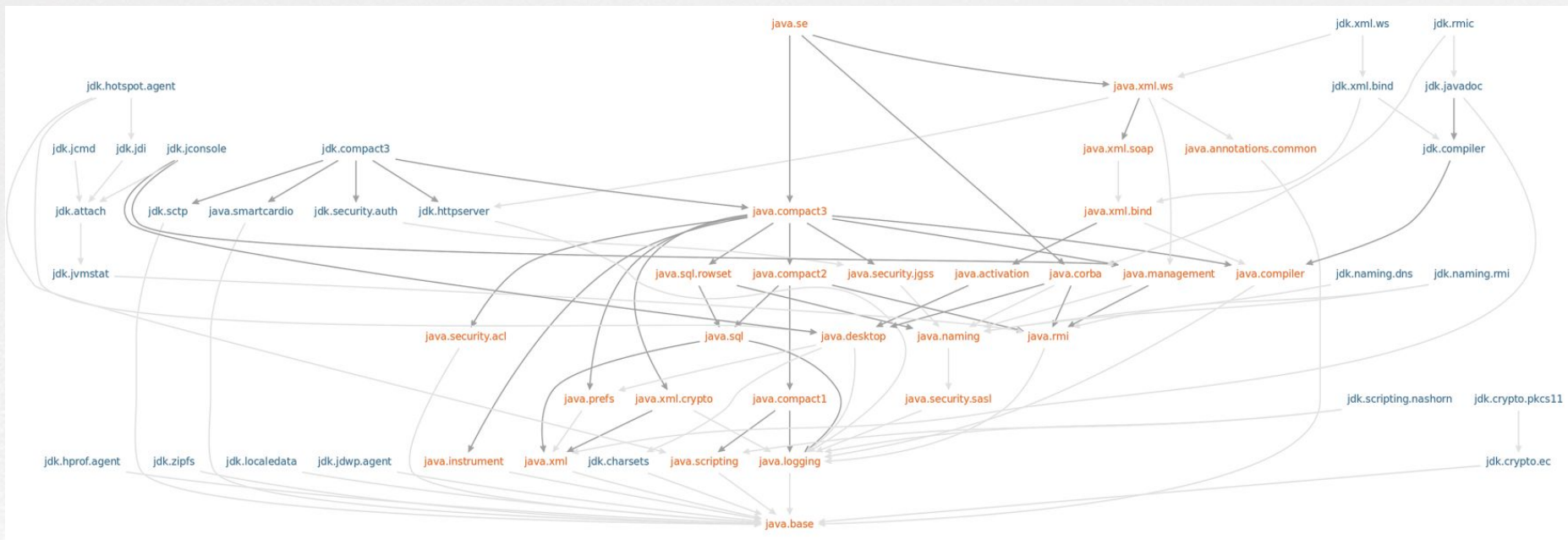
MOTIVACIÓN 2: OBTENER UN JRE MODULAR Y ADAPTADO

- Para ejecutar una aplicación java necesitamos una JVM (JRE, *Java Runtime Environment*).
- Este incluye un conjunto muy grande de clases, interfaces, ... de los cuales, puede que usemos solamente una parte.
- *rt.jar*
- Esto puede influir en el tiempo de ejecución, memoria consumida, número mayor de potenciales ataques o errores, ...

¿QUÉ OFRECE *JIGSAW*?

- **Módulo:** conjunto de paquetes, clases, interfaces, ... que permite definir cómo puede interactuar con el entorno.
- Permite definir las dependencias que va a tener con otros módulos de forma explícita.
- Encapsulación fuerte: se expone solamente una API pública (por ejemplo, mediante interfaces); las implementaciones se ocultan para que no puedan ser usadas ni accidental ni explícitamente.

JDK TAMBIÉN SE HA DESCOMPUESTO EN MÓDULOS



¿CÓMO SE DEFINE UN MÓDULO?

- A través de un fichero llamado *module-info.java*
- Debe contener
 - **Nombre**
 - **Paquetes que pone a disposición del público**
 - **Módulos de los que depende**
 - *Servicios que consume o que ofrece* (se utilizan en contextos muy concretos)

CONTENIDO DE *module-info.java*

- *exports*: indica qué paquetes van a estar disponibles fuera del módulo.

```
module otromodulo {  
    exports otra.ruta.de.paquetes;  
}
```

- *requires*: indica la dependencia con otro módulo

```
module unmodulo {  
    requires java.logging;  
    requires otromodulo;  
}
```


CONTENIDO DE *module-info.java*

- *exports package to module*: indica que un determinado paquete se exporta para ser usado en un módulo concreto.
- *requires transitive*: por defecto, no hay transitividad
 - Si $A \rightarrow B$, y $B \rightarrow C$, no tenemos directamente que $A \rightarrow C$
 - Sería muy engorroso tener que añadir siempre todas las dependencias transitivas manualmente
 - Si usamos *requires transitive* de B a C, todo el que requiera a B, tendrá acceso a C.

CONTENIDO DE *module-info.java*

- *open module*:
 - Indica que el módulo está disponible para acceder a través de introspección (API Reflection)
 - Esto es importante para librerías como Spring o Hibernate.

TIPOS DE MÓDULOS

- Automáticos
 - Permiten la compatibilidad hacia atrás.
 - Librerías que no están definidas como un módulo explícitamente.
 - Útil para poder seguir utilizando librerías que ya no tienen mantenimiento.

TIPOS DE MÓDULOS

- Con nombre
 - Definidos explícitamente con un *module-info.java*
 - Acceden a los módulos establecidos a través de *requires*
- Plataforma
 - El propio JDK modularizado
- Sin nombre
 - Formado por todas las librerías (JAR) colocadas en el classpath.

HERRAMIENTAS EN EL JDK 9 PARA LA MODULARIZACIÓN

- *jdeps*: permite saber las dependencias de un fichero jar o un módulo.
- *jlink*: ayuda a crear un entorno de ejecución a través de los módulos propios y lo de la plataforma que sean estrictamente necesarios.

JSHELL: UN REPL PARA JAVA

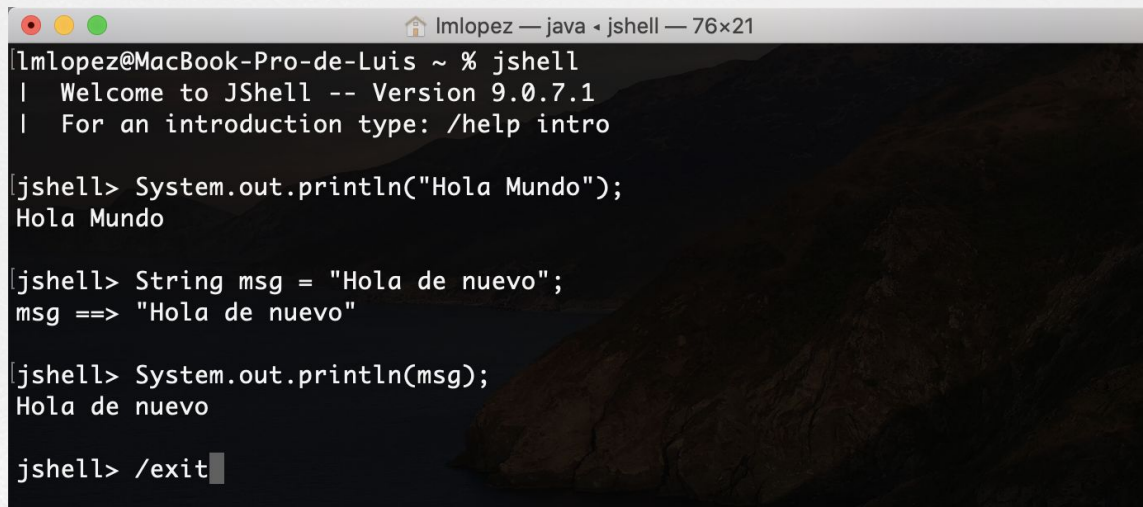
REPL: READ, EVAL, PRINT AND LOOP

- Herramienta *shell* que nos permite evaluar código sin necesidad de tener un IDE.
- Aroma a ***interpretado*** en un **lenguaje compilado**.
- Nos permite probar código sin necesidad de ***crear un Main***
- En consonancia con otros lenguajes de programación más modernos.

REPL: READ, EVAL, PRINT AND LOOP

- Ciclo $R \rightarrow E \rightarrow P \rightarrow L$
- *READ*: se lee lo que introducimos por consola
- *EVAL*: se ejecuta lo que hemos escrito
- *PRINT*: se imprime por consola el resultado
- *LOOP*: vuelta a empezar; se espera que se introduzcan nuevas sentencias por la consola.

JSHELL

A screenshot of a terminal window titled "lmlopez — java • jshell — 76x21". The terminal shows the following commands and output:

```
lmlopez@MacBook-Pro-de-Luis ~ % jshell
| Welcome to JShell -- Version 9.0.7.1
| For an introduction type: /help intro

[jshell> System.out.println("Hola Mundo");
Hola Mundo

[jshell> String msg = "Hola de nuevo";
msg ==> "Hola de nuevo"

[jshell> System.out.println(msg);
Hola de nuevo

jshell> /exit
```

<https://docs.oracle.com/javase/9/jshell/introduction-jshell.htm#JSHEL-GUID-630F27C8-1195-4989-9F6B-2C51D46F52C8>

NUEVAS FUNCIONALIDADES

MÉTODOS FACTORÍA EN LAS COLECCIONES

```
List<String> lista = List.of("Uno", "Dos", "Tres");  
Set<Integer> conjunto = Set.of(1,2,3);  
Map<String, Integer> mapa = Map.of(k1: "Uno", v1: 1, k2: "Dos", v2: 2, k3: "Tres", v3: 3);
```


NUEVOS MÉTODOS PARA STREAMS

- Se asemejan a un bucle while para mantener o eliminar elementos.
- *dropWhile*, *takeWhile*
- Ambas reciben un predicado

```
// Métodos dropWhile y takeWhile
System.out.println("\nEJEMPLO DE DROPTHILE");
Stream.of(1,2,3,4,5).dropWhile(i -> i < 3).forEach(x -> System.out.print(x + " "));

System.out.println("\nEJEMPLO DE TAKEWHILE");
Stream.of(1,2,3,4,5).takeWhile(i -> i <= 3).forEach(x -> System.out.print(x + " "));
```

NUEVOS MÉTODOS PARA STREAMS

- Nueva versión de *iterate*
- Se asemeja a un bucle for clásico
- Se añade un argumento de tipo predicado.

```
// Nueva versión de "iterate", que permite indicar a través  
// de un predicado "cuándo terminar"  
System.out.println("EJEMPLO DE ITERATE");  
Stream.iterate( seed: 0, i -> i < 10, i -> i+1)  
    .forEach(x -> System.out.print(x + ", "));
```

NUEVOS MÉTODOS PARA STREAMS

- *ofNullable*
- Hasta Java 8, un stream no puede contener valores nulos.
- En Java 9, se proporciona este método, que permite generar un stream con un elemento.

```
final String directorioDeConfiguracion =  
    Stream.of("app.config", "app.home", "user.home")  
        .flatMap(key -> Stream.ofNullable(System.getProperty(key)))  
        .findFirst()  
        .orElseThrow(IllegalStateException::new);  
System.out.println(directorioDeConfiguracion);
```

NUEVOS MÉTODOS EN OPTIONAL

- *ifPresentOrElse*
- Permite realizar una acción si el valor está presente, y otra si no lo está.

```
// Optional<String> usuario = Optional.empty();  
Optional<String> usuario = Optional.of("luismi");  
usuario.ifPresentOrElse(user -> System.out.println("El usuario logueado es " + user),  
    () -> System.out.println("Para utilizar el sistema primero debe loguearse"));
```

NUEVOS MÉTODOS EN OPTIONAL

- *or*
- Permite hacer una especie if

```
// Método or: nos permite devolver un optional si otro optional está vacío.  
System.out.println("\nEJEMPLO DE USO DE OR");  
System.out.println(usuario.or(() -> Optional.of("anonymous")).get());
```


NUEVOS MÉTODOS EN OPTIONAL

- *stream*
- Nos permite obtener un Stream a partir de un Optional
- Si el Optional está vacío, devuelve un Stream vacío

```
List<Optional<Integer>> listaOptional = List.of(
    Optional.of(1), Optional.of(2), Optional.empty(), Optional.of(4), Optional.empty()
);

List<String> listaNumerosEnBinario =
    listaOptional.stream() Stream<Optional<Integer>>
        .flatMap(Optional::stream) Stream<Integer>
        .map(Integer::toBinaryString) Stream<String>
        .collect(Collectors.toList());

System.out.println("\n"+listaNumerosEnBinario);
```

MÉTODOS PRIVADOS EN INTERFACES

- Java 8 permitía crear métodos *default* en interfaces
- ¿Qué pasa si varios métodos comparten una funcionalidad?
 - Cuando esto sucede en una clase, creamos un método privado, invocado desde el resto de métodos
 - Java 8 no permite métodos privados en interfaces
- Java 9 permite la creación de métodos privados en interfaces, orientados a esta idea.

MÉTODOS PRIVADOS EN INTERFACES

```
default void logInfo(String mensaje) {  
    log(mensaje, prefijo: "INFO");  
}  
  
default void logError(String mensaje) {  
    log(mensaje, prefijo: "ERROR");  
}  
  
default void logFatal(String mensaje) {  
    log(mensaje, prefijo: "ERROR FATAL");  
}  
  
private void log(String mensaje, String prefijo) {  
    System.out.println(prefijo.toUpperCase() + " - " + mensaje);  
}
```

OTRAS NOVEDADES

JAVADOC MEJORADO

- Adaptación a HTML5
- Estructura en módulos y paquetes
- Buscador potente
- <https://docs.oracle.com/javase/9/docs/api/overview-summary.html>

JAVADOC MEJORADO

OVERVIEW

MODULE

PACKAGE

CLASS

USE

TREE

DEPRECATED

INDEX

HELP

Java SE 9 & JDK 9

PREV

NEXT

FRAMES

NO FRAMES

ALL CLASSES

SEARCH: Collections

Java® Platform, Standard Edition & Java Development Kit Version 9 API Specification

This document is divided into three sections:

- Java SE**
 The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general use.
- JDK**
 The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in modules whose names start with `jdk`.
- JavaFX**
 The JavaFX APIs define a set of user-interface controls, graphics, media, and web packages. Names start with `javafx`.

Java SE

| Module | Description |
|--------------------------|--|
| java.activation | Defines the JavaBeans Activation Framework (JAF) API. |
| java.base | Defines the foundational APIs of the Java SE Platform. |
| java.compiler | Defines the Language Model, Annotation Processing, and Java Compiler API. |
| java.corba | Defines the Java binding of the OMG CORBA APIs, and the RMI-IIOP API. |
| java.datatransfer | Defines the API for transferring data between and within applications. |
| java.desktop | Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans. |

Packages

- `javafx.base/javafx.collections`
- `javafx.base/javafx.collections.transformation`

Types

- `java.util.Collections`
- `javafx.collections.ArrayChangeListener`
- `javafx.collections.ListChangeListener.Change`
- `javafx.collections.MapChangeListener.Change`
- `javafx.collections.SetChangeListener.Change`
- `javafx.collections.transformation.FilteredList`
- `javafx.collections.FXCollections`
- `javafx.collections.ListChangeListener`
- `javafx.collections.MapChangeListener`
- `javafx.collections.ModifiableObservableListBase`
- `javafx.collections.ObservableArray`
- `javafx.collections.ObservableArrayBase`
- `javafx.collections.ObservableFloatArray`
- `javafx.collections.ObservableIntegerArray`
- `javafx.collections.ObservableList`
- `javafx.collections.ObservableListBase`
- `javafx.collections.ObservableMap`
- `javafx.collections.ObservableSet`
- `javafx.collections.SetChangeListener`
- `javafx.collections.transformation.SortedList`
- `javafx.collections.transformation.TransformationList`
- `javafx.collections.WeakListChangeListener`

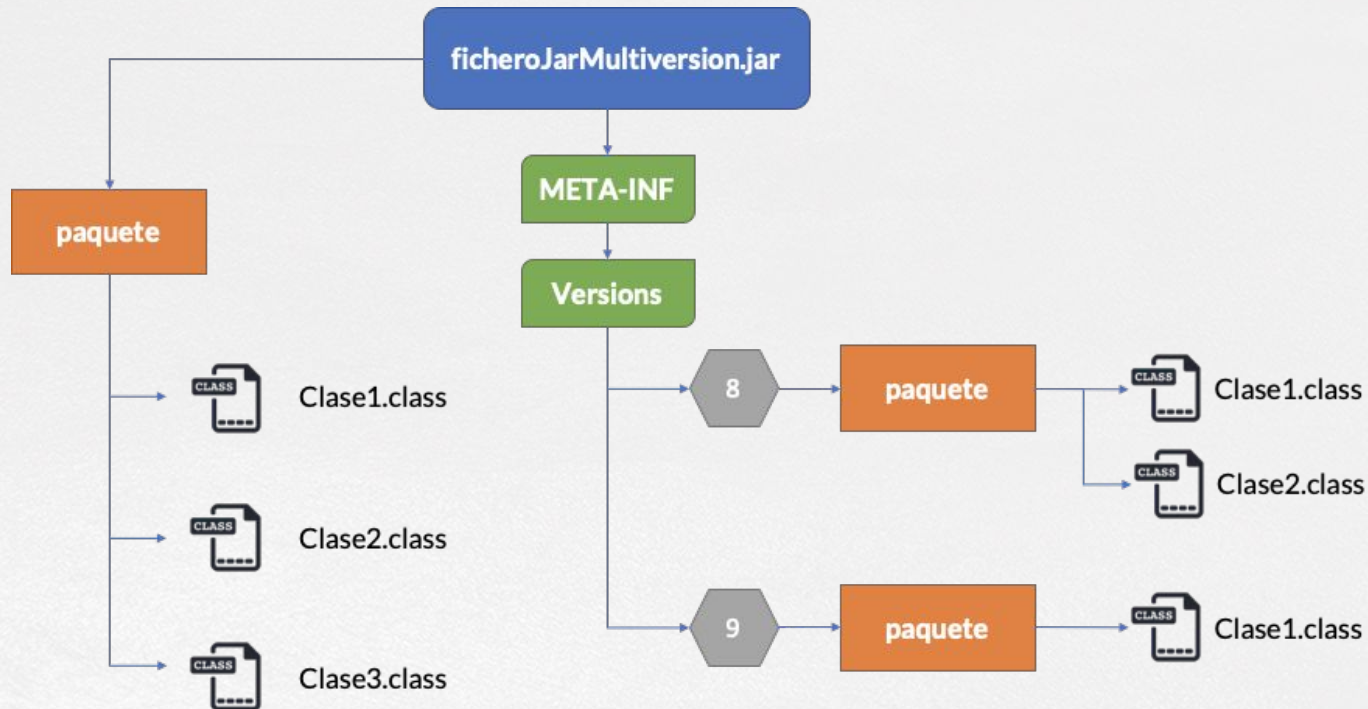
JAR MULTIVERSIÓN

- Hasta ahora, los programadores de librerías que querían dar soporte para varias versiones de Java tenían 2 opciones
 - Un JAR por versión
 - Un único JAR con la versión mínima
 - No se pueden aprovechar funcionalidades de nuevas versiones.

JAR MULTIVERSIÓN

- Con Java 9, un JAR puede incluir
 - Clases para cualquier versión de Java
 - Clases específicas para una versión.
- Esto permitiría
 - Programar una clase específica para quién utilice aún Java 8
 - Otra clase específica si alguien utiliza Java 9, aprovechando nuevas funcionalidades.

JAR MULTIVERSIÓN



TRY-WITH-RESOURCES

- Funcionalidad incluida en Java 7
- A partir de Java 9, la sintaxis permite definir los recursos fuera, siempre que sean final o efectivamente final.

```
final Resource r1 = ....;
Resource r2 = ...;
try (r1; r2) {
    //...
} catch (Exception ex) {
    //...
}
```


RECOLECTOR DE BASURA G1 MEJORADO

- Se cambia el recolector de basura por defecto al llamado G1.
- Optimizado para alto rendimiento y baja latencia.
- El recolector de basura es un proceso que se encarga de ir liberando la memoria de los objetos que ya no usamos.
- Más información en <https://docs.oracle.com/javase/9/gctuning/garbage-first-garbage-collector.htm#JSGCT-GUID-ED3AB6D3-FD9B-4447-9EDF-983ED2F7A573>

STRINGS COMPACTOS

- Hasta Java 8, un String era, internamente un *char[]*.
- En Java, cada char ocupa 16 bits
- Para la mayoría de idiomas representados con el alfabeto occidental, nos bastan 256 símbolos → 8 bits
- Java 9 representa los Strings como *byte[]*
- Mejora significativa del espacio en memoria y el rendimiento.

API APPLET DEPRECADO

- Los applets están en claro desuso
- Los navegadores tienden a no dar compatibilidad con los plugins de Java.
- Se sugiere orientar a los usuarios hacia Java Web Start
https://www.java.com/es/download/faq/java_webstart.xml

TODAS LAS FUNCIONALIDADES

- <https://openjdk.java.net/projects/jdk9/>
- En esta web se puede consultar un listado exhaustivo de todas las funcionalidades incluidas o excluidas.
- También se puede acceder a la documentación de las mismas.