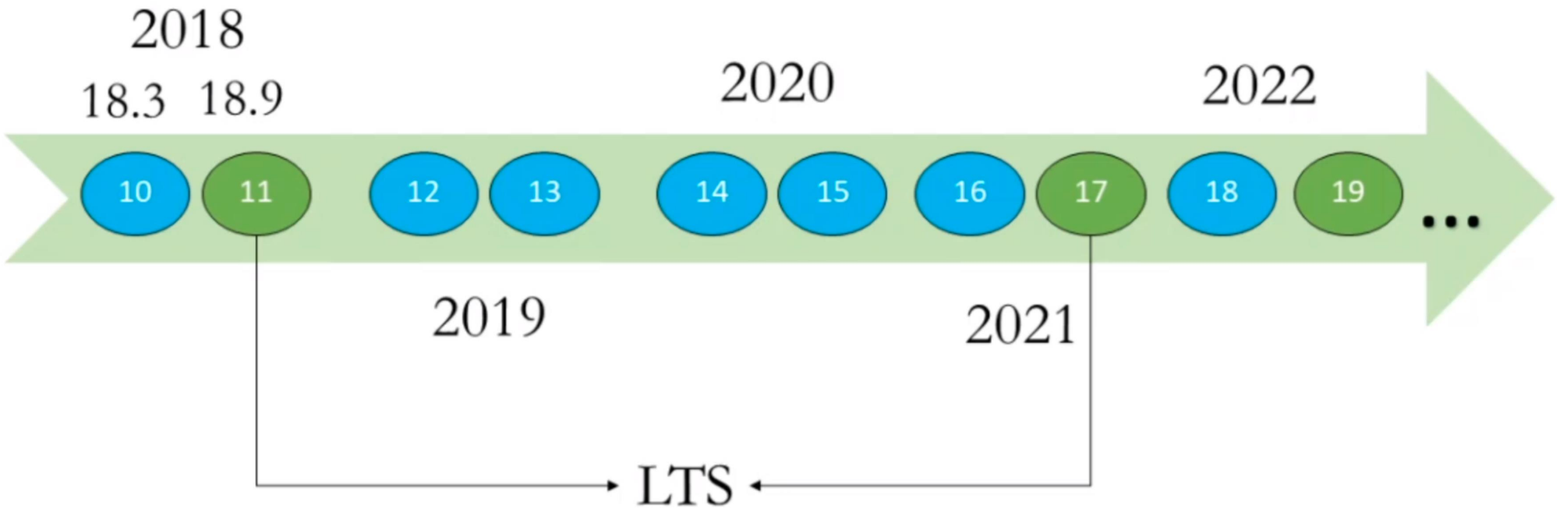

Java 8 a 17

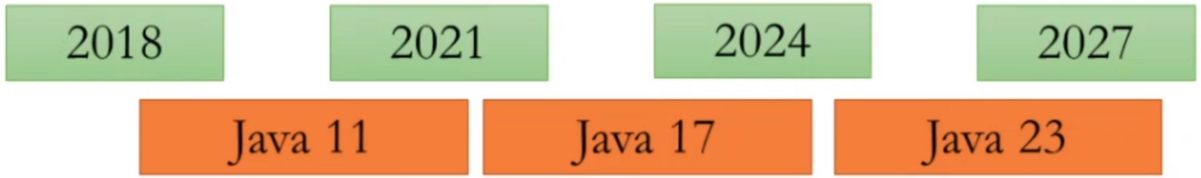
Índice

Introducción

Big Releases en Java

- Java 6 -> Diciembre 2006
- Java 7 -> Julio 2011
- Java 8 -> Marzo 2014
- Java 9 -> Septiembre 2017
- Java 10 -> Marzo 2018
- Java 11 -> Septiembre 2018
- ...
- Java 20 -> Marzo 2023





Java 8

- Programación funcional vs imperativa

- El paradigma de la *programación funcional* se ha creado explícitamente para permitir un enfoque puramente funcional de la resolución de problemas. La programación funcional es una forma de *programación declarativa*. Por el contrario, la mayoría de lenguajes más populares, incluidos los lenguajes de programación orientada a objetos (OOP) como C#, Visual Basic, C++ y Java, se han diseñado para admitir en primer lugar la programación *imperativa* (orientada a procedimientos).

- El enfoque imperativo permite al desarrollador escribir código que especifica los pasos que el equipo debe realizar para lograr el objetivo. A veces también se denomina programación *algorítmica*. Por el contrario, un enfoque funcional implica crear el problema como un conjunto de funciones que se deben ejecutar. Es necesario definir con cuidado la entrada a cada función y qué devuelve cada función. La siguiente tabla describe algunas de las diferencias generales entre estos dos enfoques.

Característica	Enfoque imperativo	Enfoque funcional
Enfoque del programador	Cómo realizar tareas (algoritmos) y cómo realizar el seguimiento de cambios de estado.	Información deseada y transformaciones necesarias
Cambios de estado	Importante	Inexistente
Orden de ejecución	Importante	Baja importancia
Control del flujo primario	Bucles, elementos condicionales y llamadas a funciones (métodos).	Llamadas a funciones, incluyendo la recursividad.
Unidad de manipulación primaria	Instancias de estructuras o clases	Funciones como recopilaciones de datos y objetos de primera clase

Java 8

- Ventajas de funciones puras

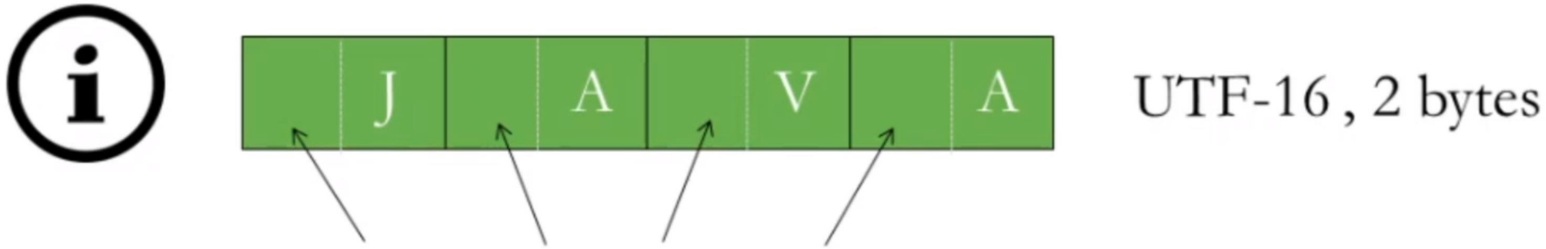
  - Mayor legibilidad y facilidad de mantenimiento. Esto se debe a que cada función está diseñada para cumplir una tarea específica dependiendo de sus argumentos. La función no depende de ningún estado externo.

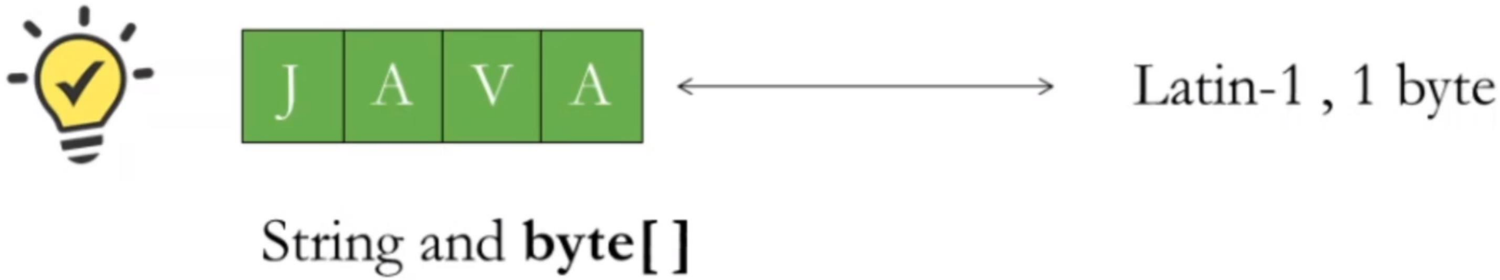
  - Desarrollo reiterativo más sencillo. Como es más sencillo refactorizar el código, la implementación de los cambios de diseño resulta a menudo más fácil.
    Al refactorizar mediante un método puro, se puede llamar al método puro cuando se desee sin preocuparse de efectos secundarios.

  - Pruebas y depuraciones más sencillas. Como las funciones puras se pueden probar más fácilmente en aislamiento, se puede escribir código de prueba que llame a la función pura con valores típicos, casos avanzados válidos y casos avanzados no válidos.

Java 9

- String compact





Java 9

- String compact

Java 9

- String compact

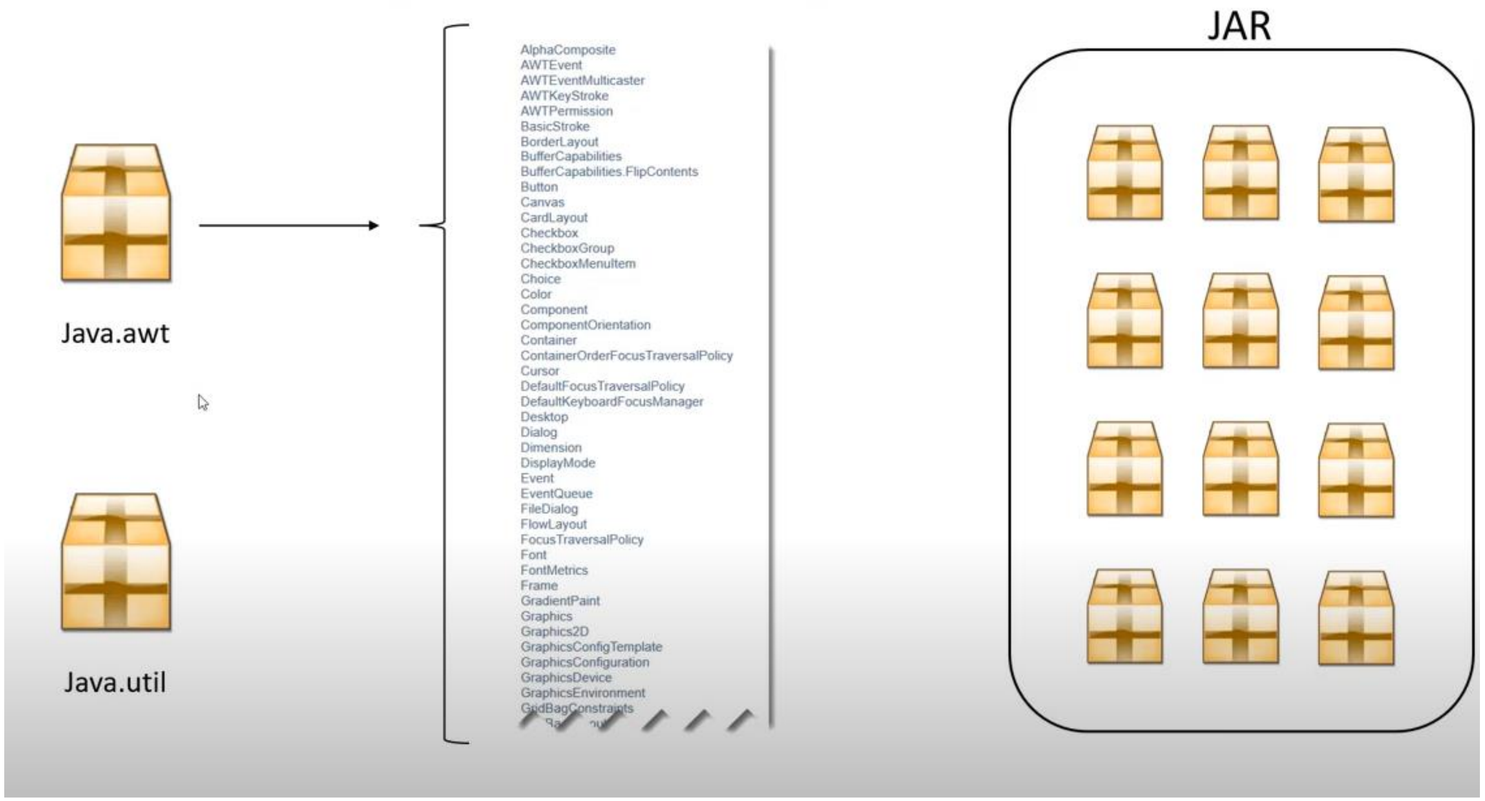
String class : JDK 8 vs. JDK 9

```
public char charAt(int index) {
    if ((index < 0) || (index >= value.length)) {
        throw new StringIndexOutOfBoundsException(index);
    }
    return value[index];
}
```

```
public char charAt(int index) {
    if (isLatin1()) {
        return StringLatin1.charAt(value, index);
    } else {
        return StringUTF16.charAt(value, index);
    }
}
```

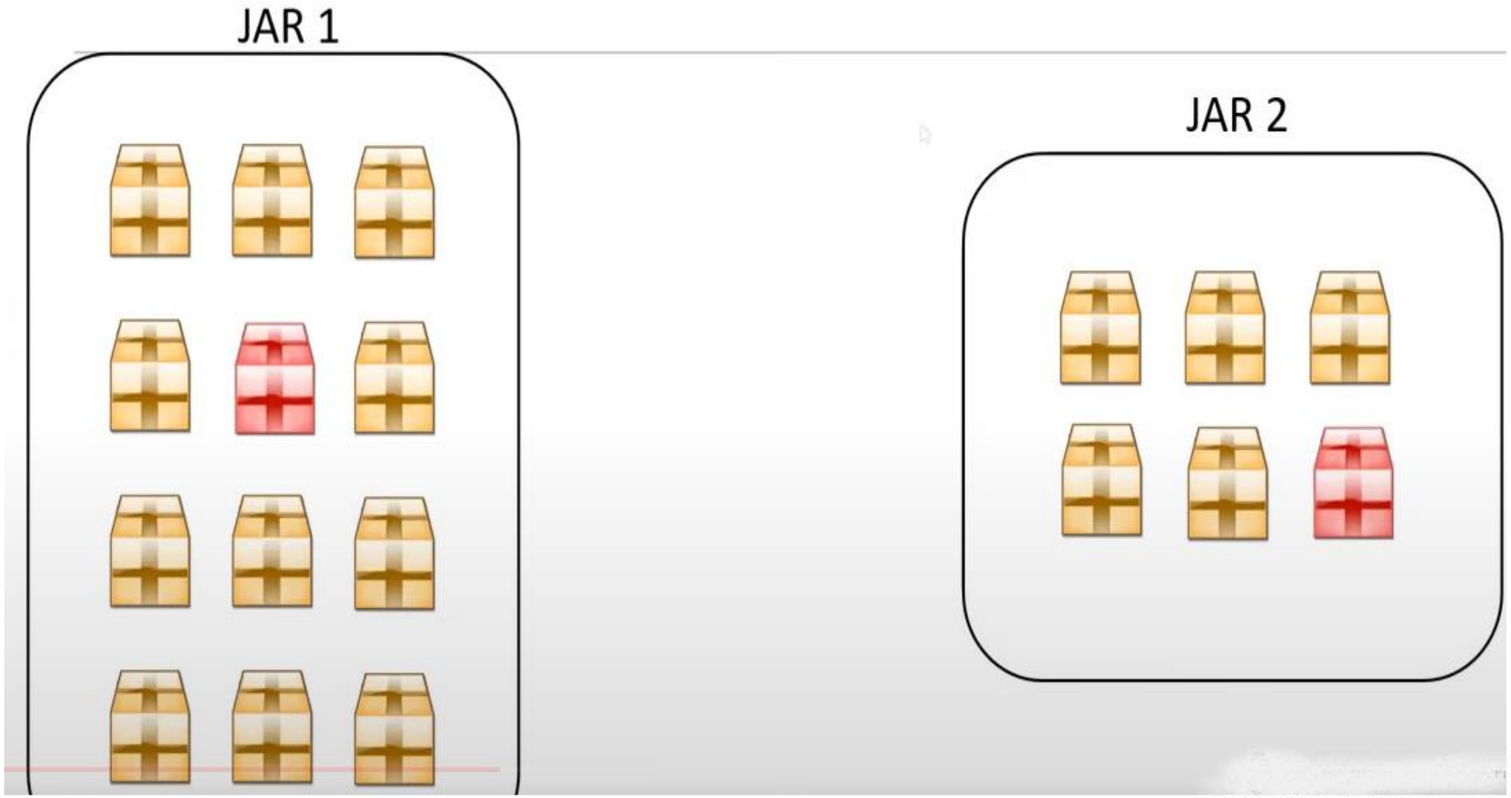
Java 9

- Módulos



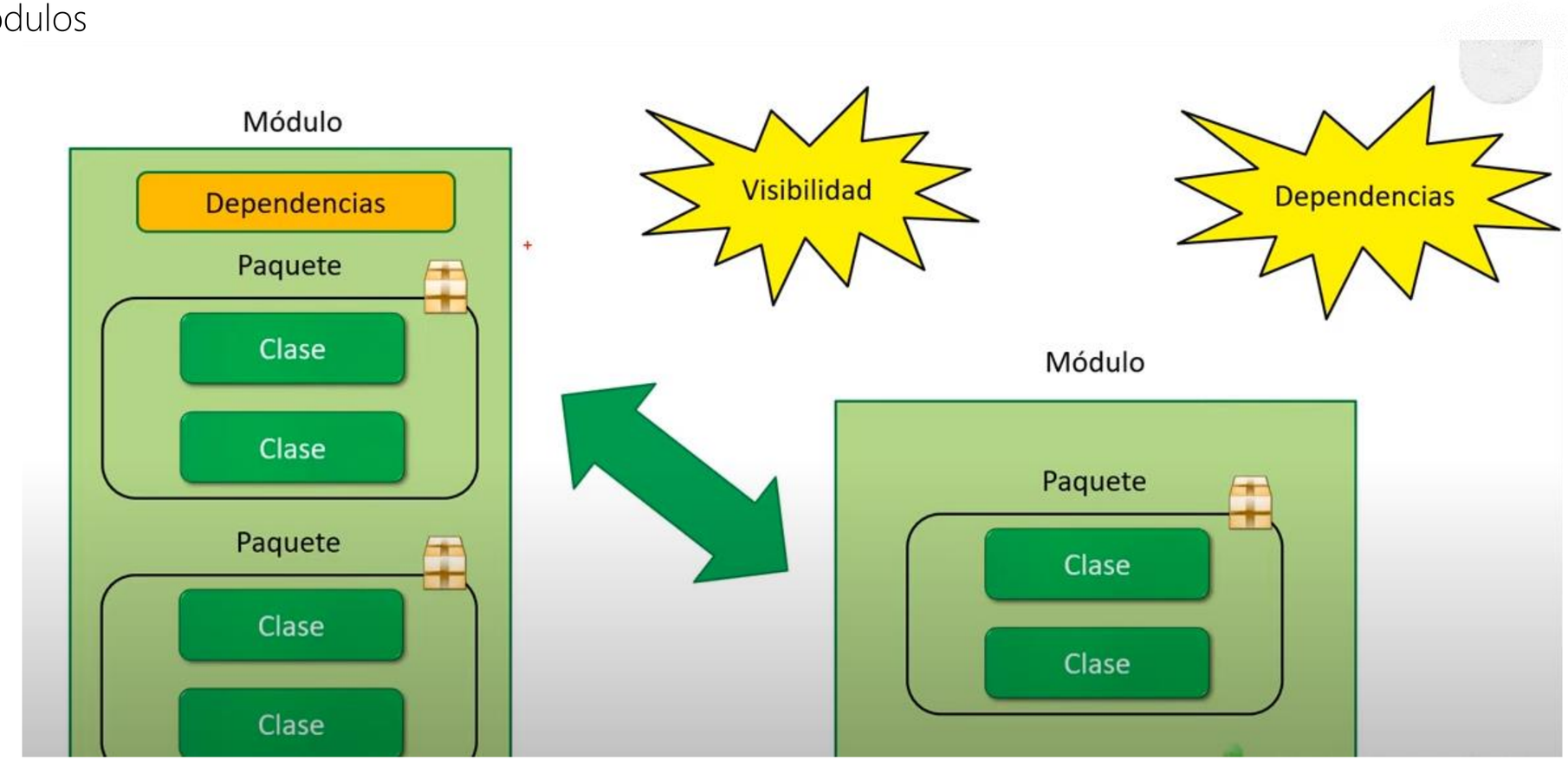
Java 9

- Módulos



Java 9

- Módulos



Java 9

- Módulos

CARÁCTERÍSTICAS

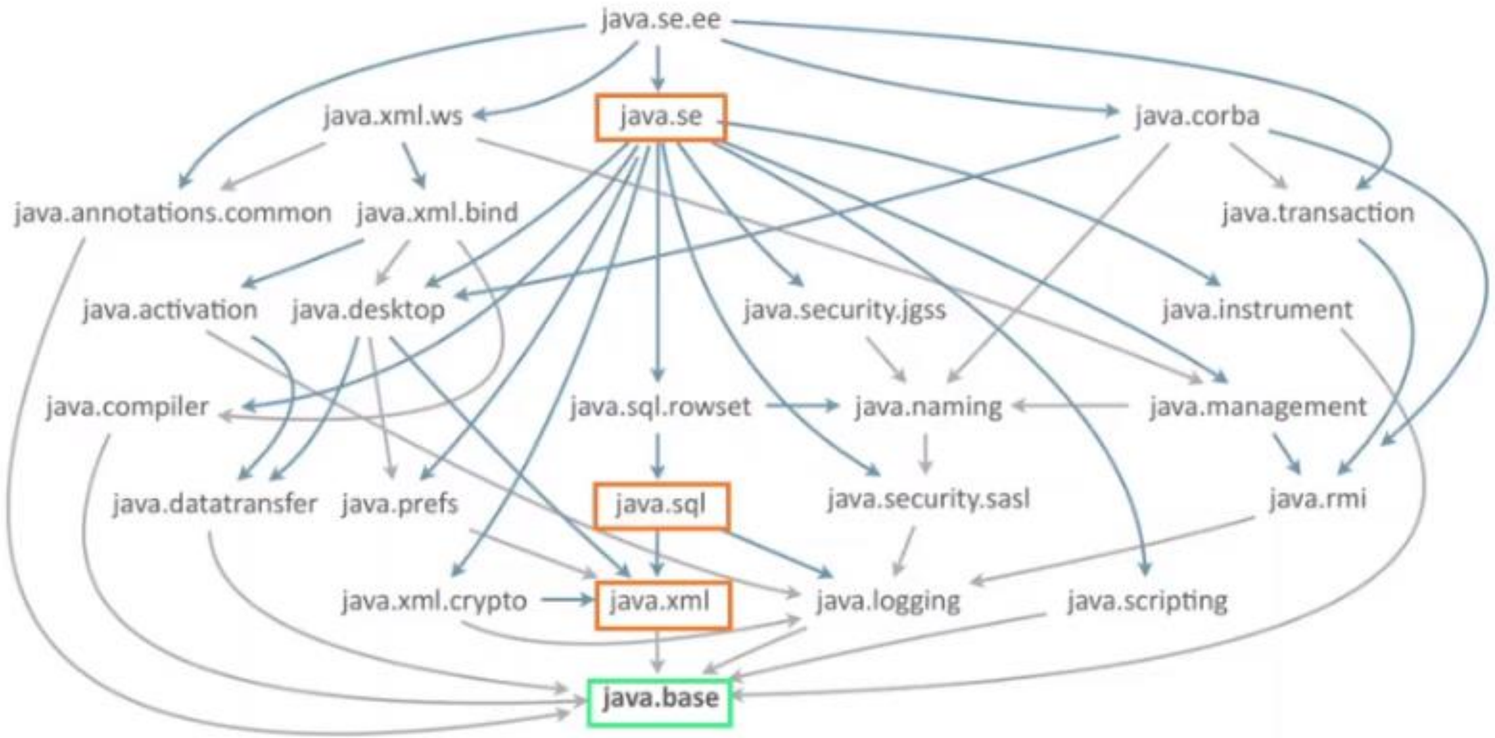
- Encapsulación fuerte
  - Especificación de parte pública y parte privada
- Interfaces de acceso bien definidas
  - La parte pública debe ser definida con cuidado y seguridad. Otros módulos pueden ser perjudicados al realizar cambios.
- Dependencias bien marcadas
  - Especifica que módulos se necesitan para el buen funcionamiento del propio módulo.

VENTAJAS

- Comprobación de dependencias antes de compilar.
- Encapsulación fuerte
  - Se evita acceso a parte privada. No dependencias sobre ella
- Seguridad
  - Limitación de objetivos de ataque
- Desarrollo diferenciado
  - Se crean límites entre los que desarrollan el módulo y los que lo usan

Java 9

- Módulos



Java 9

- Módulos

```
D:\>java --describe-module java.sql
java.sql@16.0.1
exports java.sql
exports javax.sql
requires java.transaction.xa transitive
requires java.logging transitive
requires java.base mandated
requires java.xml transitive
uses java.sql.Driver
```

Module **exports** packages

Module **requires** modules

Java 10

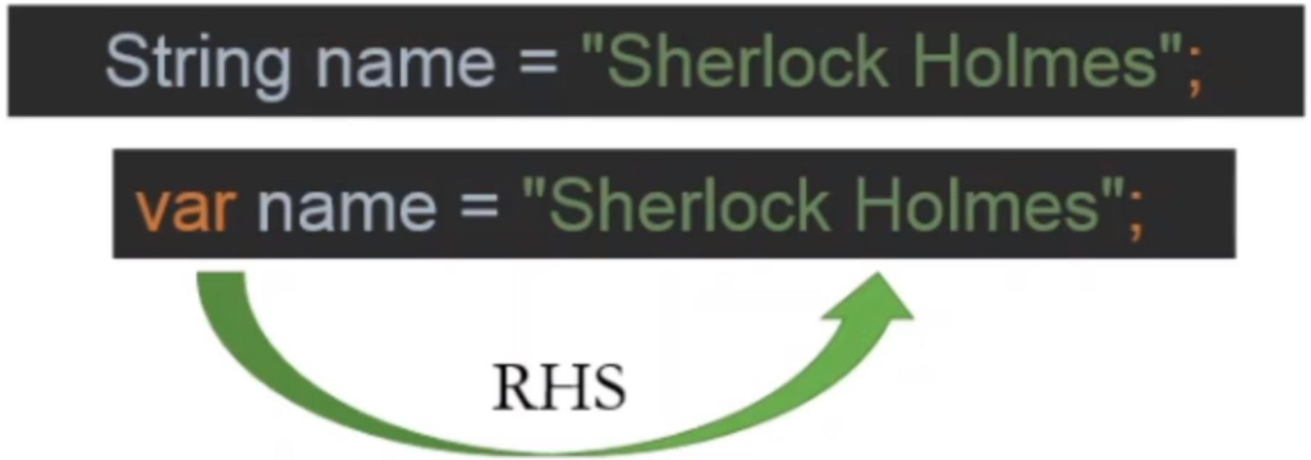
- Type inference

Java 10

- Type inference

Java 10

- Local-variable Type inference



Java 10

- Type inference

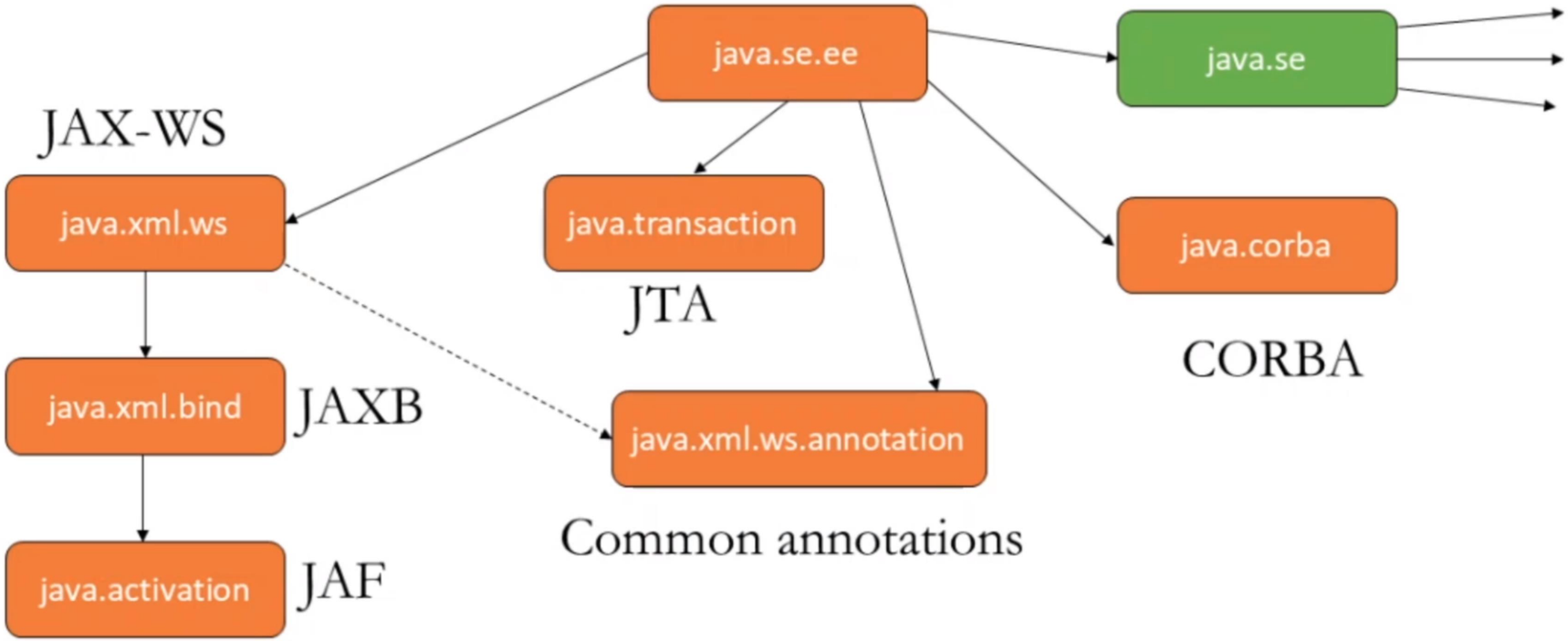
```
public static void main(String[ ] args) {
    int sum = 0;
    sum += 1;

    var sumV = 0;
    sumV += 1;
}
```

```
public static void main(java.lang.String[]);
  Code:
     0: iconst_0
     1: istore_1
     2: iinc          1, 1
     5: iconst_0
     6: istore_2
     7: iinc          2, 1
    10: return
```

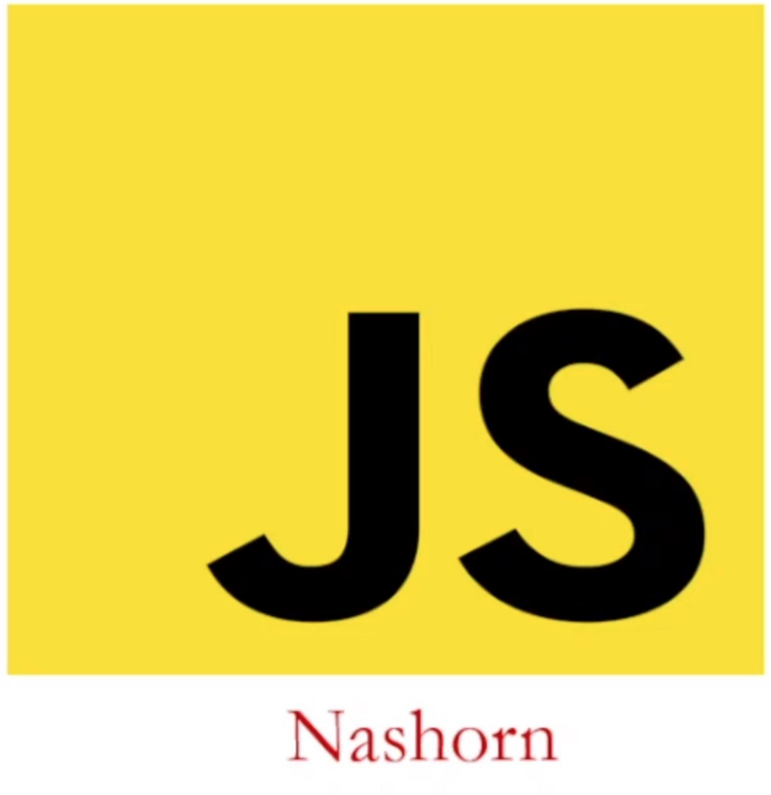
Java 11

- Deprecations/Eliminaciones



Java 11

- Deprecations/Eliminaciones



Java 11

- Deprecations/Eliminaciones

What is JavaFX ?

Oracle JDK 8,9,10

Not bundled with Java 11+

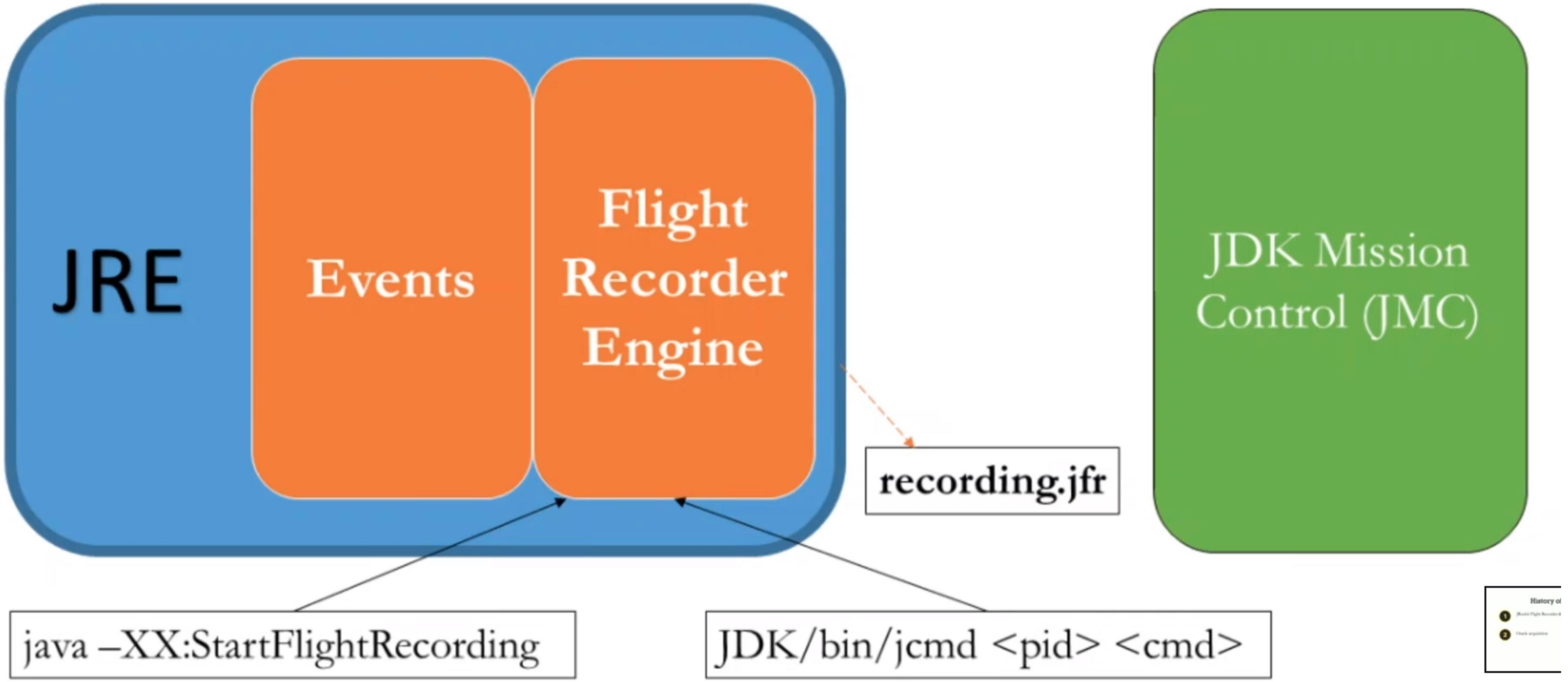
OpenJFX(openjfx.io)

JavaFX SDK

Maven/Gradle

Java 11

- JMC


JRE
Events
Flight Recorder Engine
recording.jfr
JDK Mission Control (JMC)
java –XX:StartFlightRecording
JDK/bin/jcmd <pid> <cmd>

Java 12

- Preview Features

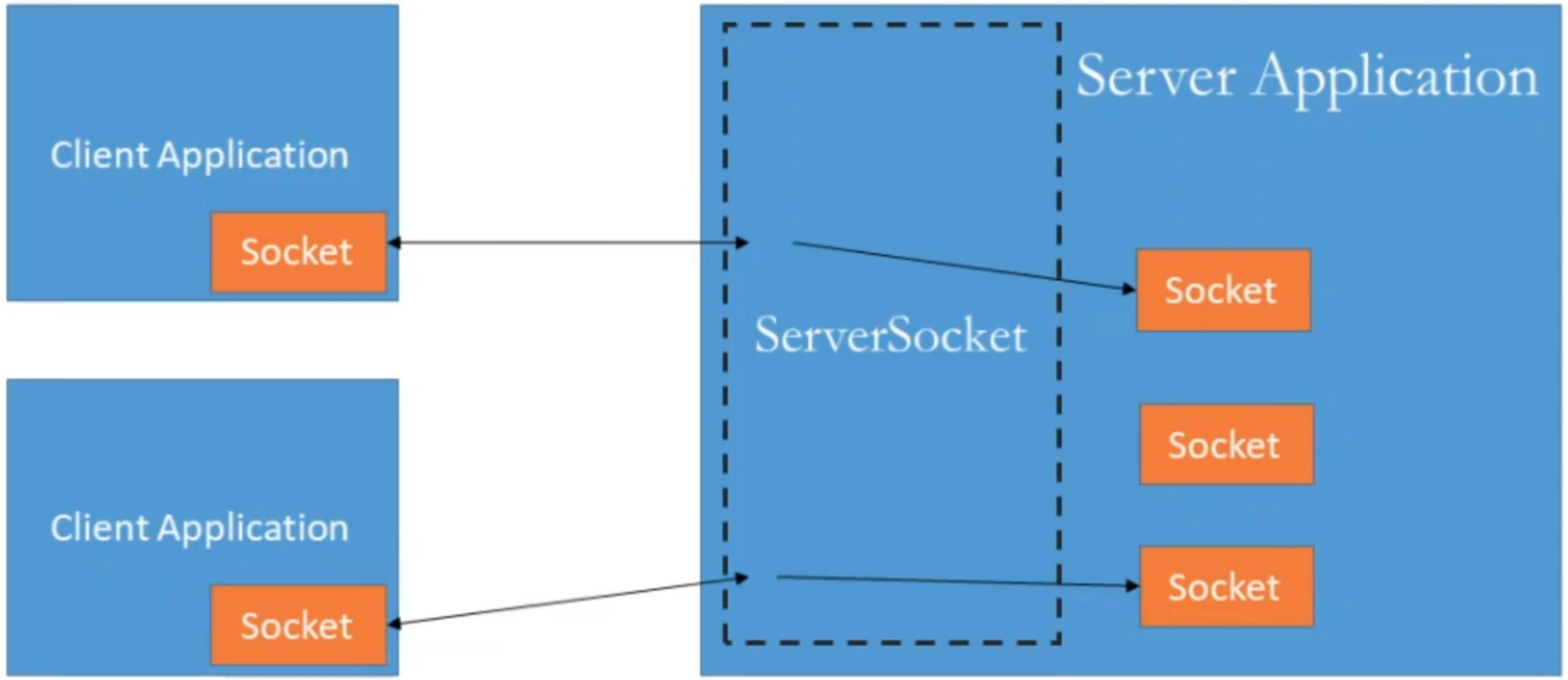
Compile : `javac --release 12 --enable-preview SwitchPreview.java`

Run : `java --enable-preview SwitchPreview`

```
var result = switch (status) {
    case ORDERED -> "Food ordered,Restaurant will confirm";
    case COOKING -> "Restaurant confirmed order,food is being cooked";
    case DELIVERED -> "Food has been delivered";
    case CANCELLED -> "Food order has been cancelled";
};
```

Java 13

- Socket API



FORMADORES { IT }

www.formadoresit.es

Consultores y formadores freelance