# Java 11 + Spring 5

– 2020 –

# Agenda

- Novedades Java 8
- Novedades Java 9
- Novedades Java 10
- Novedades Java 11
- Novedades Java 12
- Novedades Java 13
- Novedades Java 14
- Novedades Java 15
- Spring 4.3
- Spring 5.1

# Java 8

# Expresiones lambda

- Es un bloque de código
- Método anónimo
- Parecido al resultado obtenido con clases anónimas
- Termina creando una instancia de una interfaz funcional

# Expresiones lambda: Ventajas

VENTAJAS

- Código más compacto y limpio
- Da lugar al surgimiento de Streams API
- + declarativo, - implementativo
- Permite aplicar varias optimizaciones
- Inferencia de tipos
- Mismas reglas de scope que los bloques de código tradicionales
- Primer paso para la inclusión de programación funcional

# Expresiones lambda: Contras

CONTRAS

- No son funciones reales, son métodos anónimos.
- Sólo permite capturar variables inmutables y miembros de la clase.
- No se puede declarar la firma de una “función” anónima:

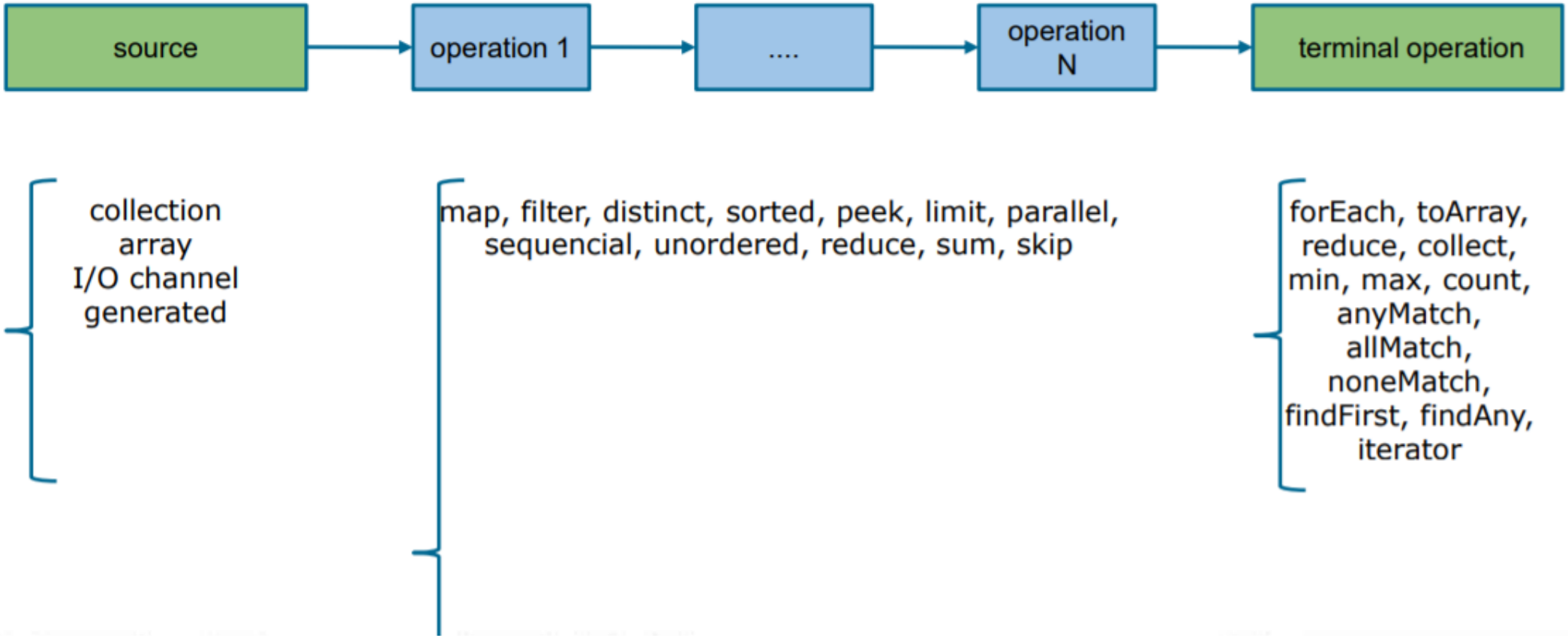
*public static void runOperation((int -> int) operation) { ... }*

# Streams

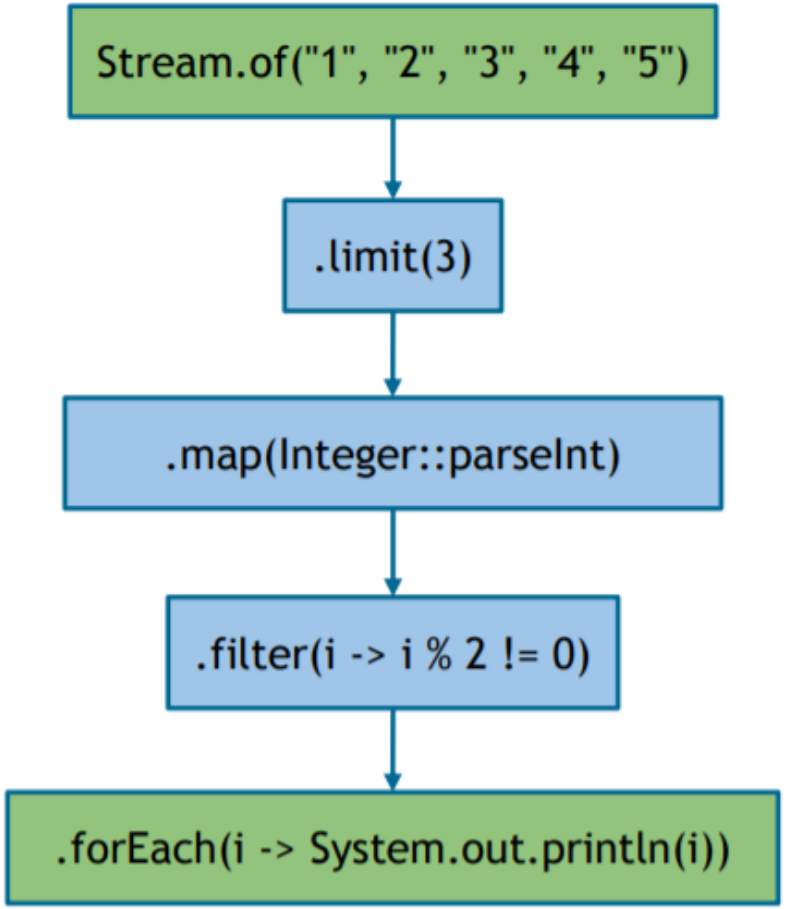
Un stream es una secuencia de elementos generada a partir de:

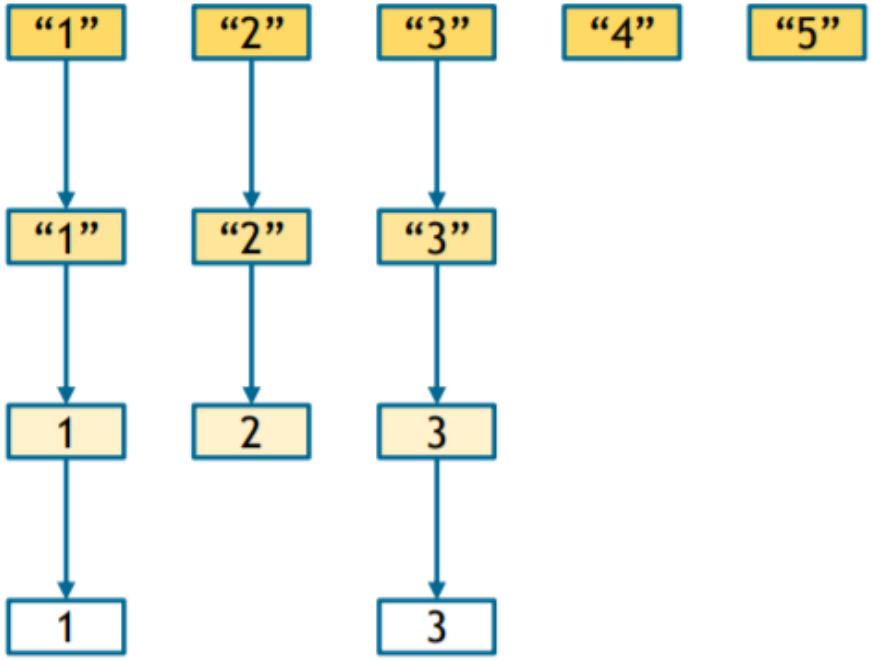
- Collections
- Arrays
- Generators (on the fly)
- I/O Channels
- Los streams no guardan los elementos
- Mecanismo de procesamiento con pipeline de operaciones.
- En el procesamiento se aplican varias técnicas de optimización.
- Las expresiones lambda le aportan mucha legibilidad.
- Provee mecanismo versátil para recolectar los resultados.

# Streams


source
operation 1
....
operation N
terminal operation
collection
array
I/O channel
generated
map, filter, distinct, sorted, peek, limit, parallel, sequencial, unordered, reduce, sum, skip
forEach, toArray, reduce, collect, min, max, count, anyMatch, allMatch, noneMatch, findFirst, findAny, iterator

# Streams


Stream.of("1", "2", "3", "4", "5")
.limit(3)
.map(Integer::parseInt)
.filter(i -> i % 2 != 0)
.forEach(i -> System.out.println(i))


“1”
“2”
“3”
“4”
“5”
“1”
“2”
“3”
1
2
3
1
3

# Streams

| Name | Returns | | Interface | Lambda signature |
|---|---|---|---|---|
| filter | Stream<T> | *lazy* | Predicate<T> | T -> boolean |
| map | Stream<U> | *lazy* | Function<T, U> | T -> U |
| sorted | Stream<T> | *lazy* | Comparator<T> | (T, T) -> int |
| limit | Stream<T> | *lazy* | n/a | n/a |
| skip | Stream<T> | *lazy* | n/a | n/a |
| reduce | T | *eager* | BinaryOperator<T> | (T, T) -> T |
| findFirst | T | *eager* | n/a | n/a |
| groupBy | Map<U, Collection<T>> | *eager* | Function<T, U> | T -> U |
| forEach | void | *eager* | Consumer<T> | T -> void |

# Interfaz funcional

- Es una Interface que tiene un solo método abstracto.
- La anotación @FunctionalInterface se puede usar como mecanismo de advertencia en la compilación.
- Se suelen implementar métodos por defecto dentro.
- Antes de crear una nueva mirar las predefinidas en java.util.function.*
- Los tipos de datos primitivos tienen interfaces funcionales predefinidas específicas.
- Son usadas en conjunto con expresiones lambdas

# Interfaz funcional: Predefinidas genéricas

- Supplier<T>: Devuelve un T
- Consumer<T>: Recibe un T
- BiConsumer<T,U>: Recibe un T y un U
- Predicate<T>: Evalúa un T y devuelve un boolean
- Function<T,R>: Recibe un T y devuelve un R
- BinaryOperator<T>: Recibe 2 T y devuelve un T

# Interfaz funcional: Métodos por defecto

- Permiten definir Interfaces que tengan algo de comportamiento.
- Son heredados por las clases implementadoras
- Al heredar más de una implementación del mismo método se puede elegir cual usar en la subclase.
- Facilitan “backward compatibility” de Collections
- Solo interactúan con otros métodos y son “stateless”.

# Fecha y hora en Java 8

- **LocalDate**, representa una fecha sin tener en cuenta el tiempo. Haciendo uso del método of(int year, int month, int dayOfMonth), se puede crear un LocalDate.
- De manera similar, se tiene **LocalTime**, la cual representa un tiempo determinado. Haciendo uso del método of(), esta clase puede crear un LocalTime teniendo en cuenta la hora y minuto; hora, minuto y segundo y finalmente hora, minuto, segundo y nanosegundo.
- **LocalDateTime**, es una clase compuesta, la cual combina las clases anteriormente mencionadas LocalDate y LocalTime.
- **Instant**, representa el número de segundos desde 1 de Enero de 1970. Es un modelo de fecha y tiempo fácil de interpretar para una máquina.
- **Duration**, hace referencia a la diferencia que existe entre dos objetos de tiempo.
- **Period**, hace referencia a la diferencia que existe entre dos fechas.

# Tipos de datos opcionales

- En Java 8 este patrón se encapsula en la clase **Optional**, la cual incluye muchos de los métodos necesarios para trabajar con este patrón.
- El uso de Java 8 **Optional** es muy práctico y refuerza los conceptos de programación funcional que tenemos en el lenguaje.
- ¿Para qué sirve un tipo **Optional**? . Su uso esta centrado en eliminar muchos de los problemas que ocurren con el manejo de excepciones de tipo *NullPointerException*.

# Motor de JS Nashorn

- Correr código dinámico Javascript nativo en la JVM
- Se basa en el uso del nuevo bytecode InvokeDynamic, presentado en Java 7
- Soporta la especificación ECMAScript 5.1
- De 2 a 10x la velocidad de Rhino
- Se acerca mucho más a la performance de V8, según algunos benchmarks llega a la mitad de velocidad.
- Tener en cuenta que V8 es exclusivo para JS.

# Java 9

# Collections

- Las Novedades en Java 9 Collections son muchas.
- Java 8 introdujo las expresiones Lambda y los Streams que fueron un salto muy importante en Java.
- Sin embargo como una tecnología nueva quedaron muchas cosas por añadir y fue únicamente un primer paso.
- Java 9 hace uso del framework de colecciones de una forma mucho más natural.
- Vamos a hablar de los nuevos métodos estáticos **of** que soportan las diferentes colecciones y que hacían mucha falta para simplificar la creación de objetos.

List.of — crea una lista

Set.of — crea un conjunto

Map.of — crea un mapa

# Interfaces

Una de las novedades que introdujo el Java 8 es la implementación default de las interfaces, sin embargo, todos los métodos tenían que ser públicos. Con la llegada de Java 9, esto cambia, y es posible definir métodos privados, por lo que solo podrán ser utilizados en dentro de la misma interface, mas precisamente, solo se podrán llamar dentro de los métodos por default. Ya que fuera de la interface no será accesibles, incluso en las clases que las implementen.

Se pueden declarar métodos privados y métodos privados estáticos.

```java
package com.centripio.hello;

public interface IHelloWorld {
    default String hello(){
        return privateHello() + privateStaticHello();
    }

    private String privateHello(){
        return "Private Hello";
    }

    private static String privateStaticHello(){
        return "Private Static Hello";
    }
}
```

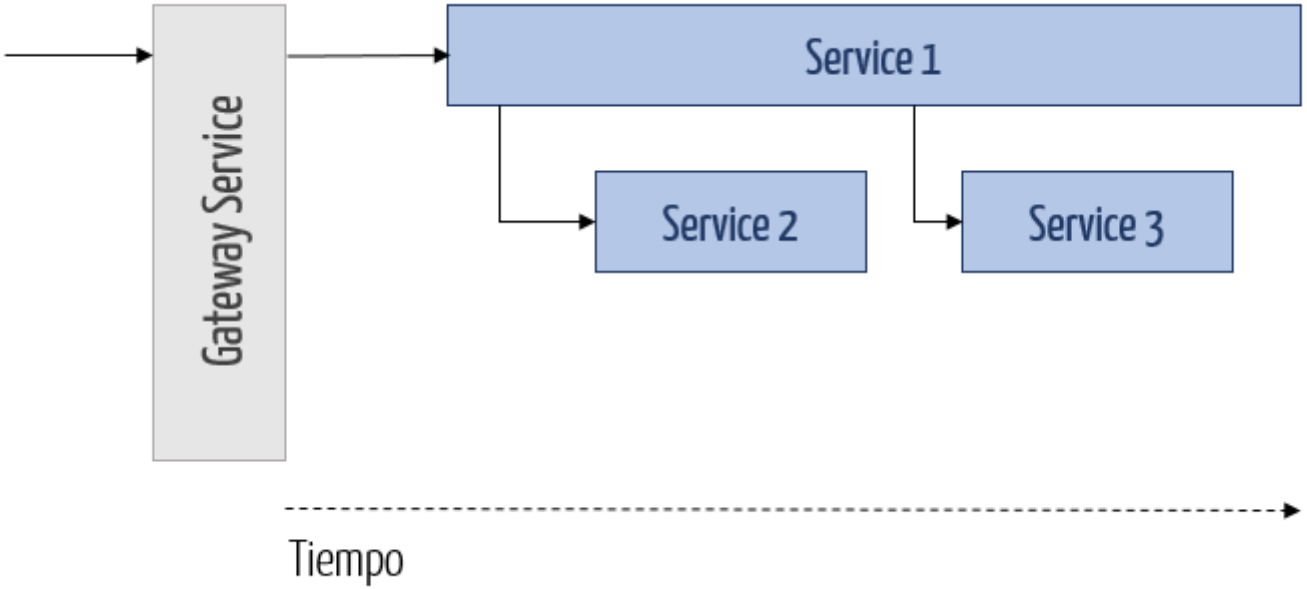
# Programación Reactiva

La programación reactiva es un paradigma enfocado en el trabajo con flujos de datos finitos o infinitos de manera asíncrona. Su concepción y evolución ha ido ligada a la publicación del **Reactive Manifesto**, que establecía las bases de los sistemas reactivos, los cuales deben ser:

- **Responsivos**: aseguran la calidad del servicio cumpliendo unos tiempos de respuesta establecidos.
- **Resilientes**: se mantienen responsivos incluso cuando se enfrentan a situaciones de error.
- **Elásticos**: se mantienen responsivos incluso ante aumentos en la carga de trabajo.
- **Orientados a mensajes**: minimizan el acoplamiento entre componentes al establecer interacciones basadas en el intercambio de mensajes de manera asíncrona.

La motivación detrás de este nuevo paradigma procede de la necesidad de responder a las limitaciones de escalado presentes en los modelos de desarrollo actuales, que se caracterizan por su desaprovechamiento del uso de la CPU debido al I/O, el sobreuso de memoria (enormes thread pools) y la ineficiencia de las interacciones bloqueantes.

# Programación Reactiva



# Modularidad

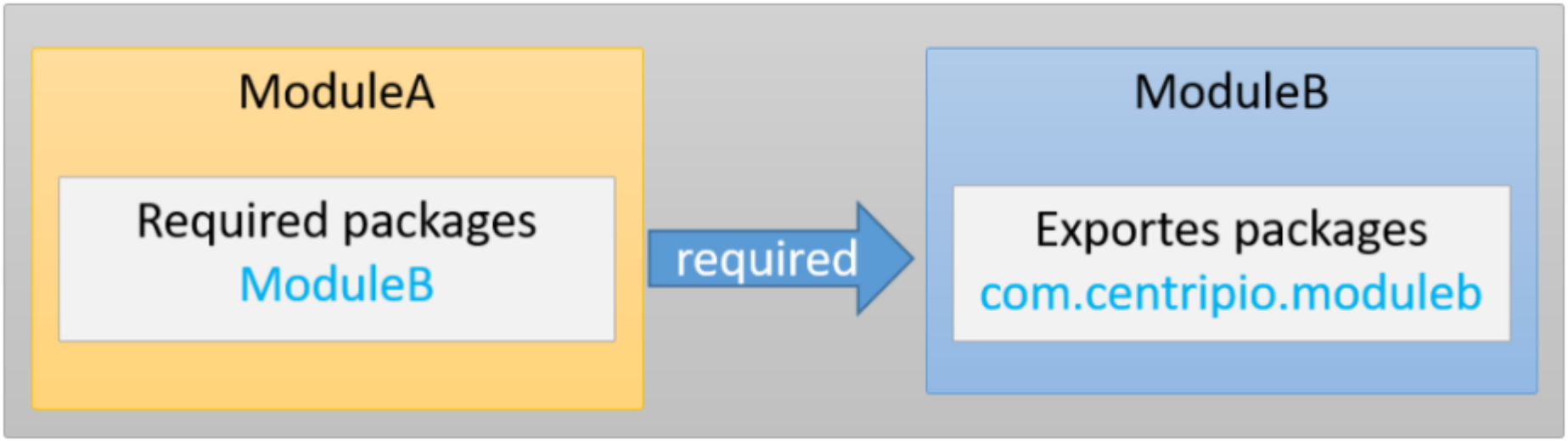
Sin lugar a duda, el sistema de módulos es la característica más relevantes y esperadas, pues desde su nacimiento, el JDK de Java ha sido monolítico, lo que implica que tenías todos o nada. En este sentido, las aplicaciones Java requerían montar en el classpath todas las clases del JDK + las clases de los proyectos. lo que implicaba varios problemas, como el espacio requerido para cargar todas las clases al classpath o la colisión en los nombres de las clases de diferentes paquetes.

El sistema de módulos ayuda a que se realice un encapsulamiento real, pues permite definir que paquetes de un módulo se expone a los demás evitando que otros módulos utilicen paquetes que no fueron expuestos, incluso mediante Reflexión.

Para crear un módulo en Java 9, es requerido que el proyecto tengo un archivo llamado module-info.java en la raíz del proyecto.

# Modularidad

El cual deberá declarar mediante la sentencia export los paquetes que podrán ser accedidos por otros módulos, y la sentencia requires para importar los módulos de los que depende. Por default, todos los paquetes que no sean expuestos serán privados de forma predeterminada.



# Cambios en la JVM 9

- Debido al declive de los Applets y que los navegadores cada vez soportan menos las tecnologías que se ejecutan con plugin como Java o Flash, Oracle ha decidido marcar todo el API de Applets como Deprecated.
- A partir de la versión Java 9, la base de datos JavaDB o más bien Apache Derby, será eliminada definitivamente del JDK, por lo que se podrá descargar por separado. En las versiones pasadas, se podía encontrar en la carpeta db del JDK.
- La herramienta VisualVM que es utilizada analizar la JVM en tiempo de ejecución fue agregada como parte del JDK en la versión 6 del JDK, pero a partir de la versión 9 de Java, es removida del JDK y se tendrá que descargar por separado. El proyecto VisualVM es mantenido por la comunidad de código abierto y puede ser descargado en: https://visualvm.github.io/

# Java 10

# Inferencia de tipos para variables locales

De las novedades la inferencia de tipos para variables locales es la más destacada en cuanto a cambios en el lenguaje con la adición de la nueva palabra reservada **var**, esto ayuda a no tener que repetir varias veces los tipos en la construcción de un objeto. En las lambdas los parámetros no es necesario declararlos infiriéndose de la interfaz que implementan. La inferencia de tipos es la idea que permite al compilador obtener el tipo estático sin que sea necesario escribirlo de forma explícita.

Java no es el único o primer lenguaje en incluir la inferencia de tipos para variables. Ha sido usado en otros lenguajes durante décadas. En realidad la inferencia de tipos incluida en Java 10 con var es muy limitada y restringida de manera intencionada. Si no fuese así el algoritmo Hindley-Milner usado para la inferencia de tipos usado en la mayoría de lenguajes que toma un tiempo exponencial en el peor de los casos potencialmente disminuiría la velocidad de javac.

La inferencia de tipos para variables locales hace que el código no sea tan verboso sin perder en gran medida la legibilidad ya que solo es para las variables locales.

# Aumento de la legibilidad

La inferencia de tipos definitivamente reduce la cantidad de tiempo para escribir código Java pero mejor es la mejora en legibilidad del código. Los desarrolladores dedican mucho más tiempo a leer código fuente que el que dedican a escribirlo de manera que definitivamente hay que optimizar para la facilidad de lectura sobre la facilidad de escritura. Aunque var no siempre es una mejora en cuanto a legibilidad ya que se pierde la información del tipo su uso se guía por el principio de no tanto para optimizar la escritura o lectura sino generalizando más para la facilidad de mantenimiento, escribir algunos tipos genéricos no triviales es complicado aún con la ayuda de asistencia de un entorno integrado de desarrollo.

No está permitido en retornos, parámetros, propiedades, variables sin inicializar, ni asignar null pero en Java 11 el uso de var se permitirá en los parámetros de una expresión lambda que será útil porque permite un parámetro formal cuyo tipo es inferido pero que además en el que se pueden usar anotaciones.

# Aumento de la legibilidad

Con la inferencia de tipos los nombres de las variables cobran mayor importancia dado que var elimina la posibilidad al lector del código adivinar la intención de una variable a partir del tipo. Ya es difícil asignar nombres adecuados ahora supondrá mayor importancia.

El tipo en las variables locales no es tan importante ya que normalmente los nombres de las variables son el del tipo. Con var se evita repetición entre el tipo y el nombre de la variable, la brevedad de var hace destacar el nombre de la variable y proporciona mayor claridad además de tener que escribir menos código repetitivo.

# Mejoras en colecciones no modificables

Java 10 añadió un nuevo método estático **copyOf** dentro de las interfaces List, Set y Map para facilitar la creación de colecciones inmutables a partir de una colección dada:

```
List.copyOf(lista);
Set.copyOf(conjunto);
Map.copyOf(mapa);
```

Y métodos en Collectors para facilitar la creación de colecciones inmutables a partir de un stream:

```
Collectors.toUnmodifiableList();
Collectors.toUnmodifiableSet();
Collectors.toUnmodifiableMap(keyMapper, valueMapper);
Collectors.toUnmodifiableMap(keyMapper, valueMapper, mergeFunction);
```

# Application Class Data Sharing

A fin de optimizar el tiempo de arranque de la máquina virtual de Java y reducir el uso de memoria se puede crear un fichero que contenga un conjunto de clases precompiladas.

Este fichero almacena las clases en un formato de representación interna que puede cargarse directamente en memoria, y que además, por ser de sólo lectura y mapeado en memoria, es automáticamente compartido por las distintas máquinas virtuales que se encuentren levantadas a un mismo tiempo. Esta opción se llama **Class Data Sharing** (CDS).

Un fichero de este tipo se crea automáticamente cuando se instala Java desde el programa de instalación en Windows con las clases básicas de Java. Pero si la instalación se realiza descomprimiendo el programa de instalación manualmente, o copiándola de un directorio ya existente, entonces el fichero hay que crearlo con el siguiente comando:

```
java -Xshare:dump
```

# Java 11

# Eliminación de módulos Java EE y CORBA

Se eliminan del JDK paquetes ya desaconsejados hace varias versiones anteriores y que no eran muy usados en cualquier caso. Estos paquetes son los de **CORBA** una forma de llamada a procedimientos remotos que se utilizó como alternativa a RMI pero que nunca tuvo un uso extendido prefiriéndose SOAP o más recientemente interfaces REST.

La lista de paquetes eliminados son los siguientes:

- *java.xml.ws (JAX-WS, plus the related technologies SAAJ and Web Services Metadata)*
- *java.xml.bind (JAXB)*
- *java.activation (JAF)*
- *java.xml.ws.annotation (Common Annotations)*
- *java.corba (CORBA)*
- *java.transaction (JTA)*
- *java.se.ee (Aggregator module for the six modules above)*
- *jdk.xml.ws (Tools for JAX-WS)*
- *jdk.xml.bind (Tools for JAXB)*

# Sintaxis de variables locales para parámetros en lambdas

Ahora los parámetros de una lambda pueden declararse con var con inferencia de tipos. Esto proporciona uniformidad en el lenguaje al declarar los parámetros permite usar anotaciones en los parámetros de la función lambda como @NotNull.

Esta funcionalidad tiene algunas restricciones. No se puede mezclar el uso y no uso de var y no se puede mezclar el uso de var y tipos en lambdas explícitas. Son consideradas ilegales por el compilador y producirá un error en tiempo de compilación.

# Cliente HTTP

- En Java 9 se incorporó de forma experimental un cliente HTTP con soporte para HTTP/2 en el propio JDK.
- En Java 11 alcanza la categoría de estable.
- Este cliente HTTP es una forma sencilla de hacer llamadas a servicios web ya sean REST o GraphQL.
- Las clases del nuevo cliente se encuentran en el paquete java.net.http.
- Al estar este cliente HTTP incorporado en el JDK no será necesario depender de librerías de terceros.

# Ejecución desde archivo de código fuente único

- Para ejecutar un programa Java es necesario compilarlo a bytecode y posteriormente ejecutarlos. Se necesitan dos pasos.
- Para facilitar la ejecución de los programas que se componen de un único archivo de código fuente se añade la posibilidad de lanzar un programa desde el archivo de código fuente.
- Esto es útil par programas pequeños o para los casos de estar aprendiendo el lenguaje.

# Ejecución desde archivo de código fuente único

```
~/Downloads cat HelloWorld.java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
~/Downloads java HelloWorld.java
Hello, World
~/Downloads cat HelloWorld.sh
#!java --source 11

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
~/Downloads ./HelloWorld.sh
Hello, World
~/Downloads
```

# Unicode 10

Tratar texto es una parte importante casi de cualquier aplicación, este soporte de Unicode 10 añade 16018 nuevos caracteres soportados, 128 nuevos emojis y 19 símbolos nuevos para el estándar en televisiones 4K.

# TLS 1.3

Algunas de las versiones anteriores de TLS ya no se consideran seguras añadir soporte para la versión más reciente TLS 1.3 permite a las aplicaciones Java ser más compatibles y más seguras. El protocolo TLS 1.3 proporciona dos beneficios principalmente, es más seguro y más rápido que sus versiones anteriores.

# Nuevos métodos clase String

- Se añaden varios métodos a la clase String muy utilizados con los que no será necesario recurrir a librerías de terceros.
- Estos métodos son **repeat, isBlank, strip, stripLeading, stripTrailing, lines**.
- En otras clases de la API también se han añadido nuevos métodos.

# Nuevo colector de Basura ZGC y otras

- Se añaden **implementaciones** específicas para la **arquitectura Aarch64** para un mejor rendimiento en la **plataforma ARM** en algunas funciones.
- Se añade de forma experimental un **nuevo recolector de basura ZGC** con pausas para recolectar basura menores capaz de manejar cantidades de memoria pequeñas de megabytes y grandes de terabytes.
- Al mismo tiempo se añade un **recolector de basura Epsilon** que no reclama la memoria.
- Se añade **soporte para los algoritmos criptográficos ChaCha20 y Poly1305** junto con otras funcionalidades criptográficas RSASSA-PSS, AES Encryption with HMAC-SHA2 for Kerberos 5, …

# Java 12

# Evolución de la sentencia switch

- Al igual que en versiones anteriores, la 12 presenta una mejora funcional, aunque en modo experimental, para la sentencia switch.
- La nueva sintaxis permite por un lado la incorporación de lambas en los cases y por otro lado la posibilidad de retornar un valor.

# Mejoras en el Garbagge Colector

- Restauración inmediata al sistema operativo de la memoria inactiva contenida en el G1.
- **Shenandoah**. Nuevo algoritmo, en modo experimental, que optimiza los tiempos de pausa del propio GC, basado en concurrencia de hilos y que aumentando el rendimiento disminuye el tiempo de evacuación de memoria.
- Mejora del G1 en la gestión de las colecciones mixtas compuestas de regiones jóvenes y viejas. Estas últimas se considerarán opcionales y solo se recolectarán si existe algún tiempo remanente sobre el total de la pausa máxima establecida.

# Suite para Microbenchmarks

- Se potencia la medida del rendimiento de las aplicaciones mediante JMH con la inclusión de una suite que facilita la creación y modificación de benchmarks de manera sencilla a parte de aportar un catálogo de benchmarks ya construidos para su uso.

# Archivos CDS por defecto

- A diferencia de las versiones anteriores ya no es necesario cargar manualmente «class data sharing». Las aplicaciones java en esta versión comparten por defecto el archivo classes.jsa entre las diferentes JVM’s en ejecución mejorando la compilación el consumo de memoria y el tiempo de arranque.

# API de constantes en la JVM

- Supone una mejora en la gestión de las constantes de carga pertenecientes a la pila de constantes , que mediante introspección podrán ser manipuladas a partir de su descripción nominal. El principal interés se encuentra en las constantes de carga de tiempo de ejecución que correspondan a clases o métodos.

# Java 13

# Bloques de texto

- Java 13 ya ofrece soporte para strings multilínea. Esta nueva característica está disponible en preview para que se pueda probar y poder tener una implementación final en la próxima versión de Java.

# Expresiones switch mejoradas

- En Java 13 en vez únicamente el valor a retornar se permite crear bloques de sentencias para cada rama case y retornar el valor con la palabra reservada yield. En los bloques de sentencias puede haber algún cálculo más complejo que directamente retornar el valor deseado.

# Otras características incorporadas y cambios

- Las otras tres características destacadas Dynamic CDS Archives, ZGC: Uncommit Unused Memory para la mejora del recolector de basura y Reimplement the Legacy Socket API reescribiendo el código para los sockets en lenguaje Java son cambios internos que afectan a la plataforma Java pero no al lenguaje.

# Java 14

# Excepciones NullPointerException más útiles

- A partir de Java 14 las excepciones NullPointerException son más útiles e indican de forma precisa cual es el miembro de la línea de código que ha producido la excepción.

# Records

- Esta es la característica más destacada añadida al lenguaje que permite reducir significativamente el código necesario para algunas clases.
- Los registros son clases que no contienen más datos que los públicos declarados. Evitan mucho del código que es necesario en Java para definir los constructores, los métodos getter, los setter e implementar de forma correcta los métodos equals y hashCode.

# Records

- Para reducir el código de las clases de datos los registros adquieren automáticamente varios miembros:
  - Un campo privado y final para cada componente del estado en la descripción.
  - Un método de acceso de lectura para cada componente del estado de la descripción, con el mismo nombre y tipo.
  - Un constructor público cuya firma es la misma que el estado de la descripción que inicializa cada campo de su correspondiente argumento.
  - Una implementación de equals y hashCode de tal forma que dos registros son iguales si son del mismo tipo y contienen el mismo estado.
  - Una implementación de toString que incluye una representación de todos los componentes del registro con sus nombres.

# Records

- Los registros tienen algunas restricciones:
  - No pueden extender ninguna otra clase y no pueden declarar campos que no sean los privados automáticos que corresponden a los componentes de la descripción del estado en la descripción. Cualquier otro campo debe ser declarado como static. estas restricciones aseguran que la descripción del estado define su representación.
  - Los registros son implícitamente final y no pueden ser abstract. Esto significa que no pueden ser mejorados por otra clase o registro.
  - Los componentes de un registro son implícitamente final. Esta restricción hace que sean inmutables.

# ZGC para Windows y macOS

- La versión del recolector de basura ZGC que permite pausas muy reducidas en memorias de unos pocos MB hasta varios TB ahora es posible utilizarla en los sistemas operativos macOS y Windows.

# Pattern Matching para el operador instanceof

- Ahora el operador instanceof permite renombrar la variable y dentro de la rama usarla sin necesidad de realizar el cast, esto simplifica el código y evita posibles errores.

# Expresiones switch

- La características de expresiones switch introducida en modo vista previa en las versiones de Java 12 y 13 se califica como estándar.

# Java 15

# Algoritmo de firma digital Edwards-Curve (EdDSA)

- El algoritmo de firma digital EdDSA o Edwards-Curve Digital Signature Algorithm (EdDSA) es demandado por mejorar la seguridad y el rendimiento comparado con otros algoritmos de firma, ya está implementado en otras librerías de criptografía como OpenSSL. Este esquema de firma es opcional en TLS 1.3 pero es uno de los tres permitidos. Añadir este algoritmo permite usar EdDSA en Java sin recurrir a librerías de terceras partes.

# Sealed Classes (vista previa)

- Las clases sealed especifican de forma explícita que clases tiene permitido la extensión y herencia. Las clases sealed son más restrictivas que el comportamiento por defecto de permitir a cualquier clase la extensión pero más permisivo que si se utiliza la palabra clave final que impide a cualquier clase la extensión.

- Se introduce una nueva palabra reservada sealed. La declaración de la clase sealed se realiza con el siguiente sintaxis, en este ejemplo la clase Shape solo puede ser extendida por las clases Circle, Rectangle y Square.

# Clases ocultas

- Se añaden clases ocultas o hidden classes que son clases que no pueden usarse directamente por otras clases. Su intención es que sean usadas por frameworks que generan clases en tiempo de ejecución y las usan de forma indirecta con reflection.

# Reimplementación de la antigua API DatagramSocket

- Se reemplazan las implementaciones de bajo nivel para la comunicación por red java.net.DatagramSocket y java.net.MulticastSocket con una implementación mas simple y moderna que es más fácil de mantener, depurar y fácil de adaptar a los threads virtuales del proyecto Loom.

# Otras características incorporadas y cambios

- Otras especificaciones que no tienen tanto impacto desde el punto de vista del programador y en el lenguaje son las siguientes. Algunas eliminan y marcan como desaconsejado su uso.

- Entre las más destacables está Foreign-Memory Access que permite a los programas Java acceder de forma segura y eficiente a memoria externa fuera de la memoria heap de Java. También a destacar el soporte de Unicode 13.0 que añade unos 5K nuevos caracteres o el soporte para el algoritmo de hash SHA-3 en el apartado de seguridad.

# Spring 4.3

# Inyección por constructor implícita

Spring 4.3 se realizará inyección implícita en escenarios de un solo constructor, haciendo que su código sea más independiente de Spring al no requerir un @Autowired anotación en absoluto.

# Soporte de métodos de interfaz por defecto de Java 8

- A partir de 4.2, las anotaciones Spring también pueden procesarse cuando se usan en los métodos predeterminados de Java 8.
- Esto permite una gran flexibilidad en la creación de implementaciones de bean complejas y clases de configuración (la clase con **@Configuration**).
- Incluso podemos aprovechar el patrón de comportamiento de herencia múltiple del método predeterminado de Java 8.

# Resolución de dependencias mejorada

La versión más reciente también presenta **ObjectProvider**, una extensión de la interfaz existente de **ObjectFactory** con firmas útiles como **getIfAvailable** y **getIfUnique** para recuperar un bean solo si existe o si se puede determinar un solo candidato (en particular: un candidato principal en caso de múltiples beans posibles).

Puede usar dicho controlador ObjectProvider para propósitos de resolución personalizada durante la inicialización, o puede almacenar el controlador en un campo para una resolución tardía bajo demanda (como se suele hacer con un ObjectFactory).

# Mejoras en la abstracción de cache

- La abstracción del caché se utiliza principalmente para almacenar en caché los valores que consumen CPU.
- En casos de uso particulares, una clave dada puede ser solicitada por varios subprocesos (es decir, clientes) en paralelo, especialmente en el inicio.
- El soporte de caché sincronizado es una característica solicitada desde hace tiempo que ahora se ha implementado.

```
@Service
public class FooService {

    @Cacheable(cacheNames = "foos", sync = true)
    public Foo getFoo(String id) { ... }

}
```

- Observe el atributo *sync* = *true* que le indica al marco que bloquee cualquier subproceso simultáneo mientras se calcula el valor. Esto asegurará que esta operación intensiva se invoque solo una vez en caso de acceso concurrente.

# Mejoras en la abstracción de cache

- Spring 4.3 también mejora la abstracción de caché de la siguiente manera:

  - Las expresiones *SpEL* en las anotaciones relacionadas con el caché ahora pueden referirse a beans (es decir, **@beanName.method()**).
  - **ConcurrentMapCacheManager** y **ConcurrentMapCache** ahora admiten la serialización de entradas de caché a través de un nuevo atributo **storeByValue**.
  - **@Cacheable**, **@CacheEvict**, **@CachePut** y **@Caching** ahora se pueden usar como meta-anotaciones para crear anotaciones compuestas personalizadas con anulaciones de atributos.

# Variantes de RequestMappings compuestos

Spring Framework 4.3 introduce las siguientes variantes compuestas a nivel de método de la anotación **@RequestMapping** que ayudan a simplificar las asignaciones para métodos HTTP comunes y expresan mejor la semántica del método del controlador anotado.

- *@GetMapping*
- *@PostMapping*
- *@PutMapping*
- *@DeleteMapping*
- *@PatchMapping*

Por ejemplo, @GetMapping es una forma más corta de decir @RequestMapping (method = RequestMethod.GET). El siguiente ejemplo muestra un controlador MVC que se ha simplificado con una anotación compuesta de @GetMapping.

# Anotaciones @RequestScope, @SessionScope y @ApplicationScope

Cuando se usan componentes controlados por anotación o configuración de Java, se pueden usar las anotaciones **@RequestScope**, **@SessionScope** y **@ApplicationScope** para asignar un componente al ámbito requerido. Estas anotaciones no solo establecen el alcance del bean, sino que también establecen el modo de proxy con alcance en *ScopedProxyMode.TARGET_CLASS*.

El modo TARGET_CLASS significa que el proxy CGLIB se utilizará para la asignación de este bean y garantizar que se pueda inyectar en cualquier otro bean, incluso con un alcance más amplio. El modo TARGET_CLASS permite el proxy no solo para interfaces sino también para clases.

# Anotaciones @RequestAttribute y @SessionAttribute

Aparecieron dos anotaciones más para inyectar parámetros de la solicitud HTTP en los métodos del Controlador, a saber, **@RequestAttribute** y **@SessionAttribute**.
Le permiten acceder a algunos atributos preexistentes, administrados globalmente (es decir, fuera del Controlador).
Los valores para estos atributos pueden ser proporcionados, por ejemplo, por instancias registradas de *javax.servlet.Filter* u *org.springframework.web.servlet.HandlerInterceptor*.

Supongamos que hemos registrado la siguiente implementación de *HandlerInterceptor* que analiza la solicitud y agrega un parámetro de inicio de sesión a la sesión y otro parámetro de consulta a una solicitud:

```
public class ParamInterceptor extends HandlerInterceptorAdapter {

    @Override
    public boolean preHandle(HttpServletRequest request,
      HttpServletResponse response, Object handler) throws Exception {
        request.getSession().setAttribute("login", "john");
        request.setAttribute("query", "invoices");
        return super.preHandle(request, response, handler);
    }

}
```

# Anotaciones @RequestAttribute y @SessionAttribute

Dichos parámetros se pueden inyectar en una instancia de *Controller* con las anotaciones correspondientes en los argumentos del método:

```
@GetMapping
public String get(@SessionAttribute String login,
  @RequestAttribute String query) {
    return String.format("login = %s, query = %s", login, query);
}
```

# Librerías

Spring 4.3 admite las siguientes versiones de biblioteca y generaciones de servidores:

- Hibernate ORM 5.2 (todavía compatible con 4.2 / 4.3 y 5.0 / 5.1 también, con 3.6 en desuso ahora)
- Jackson 2.8 (mínimo elevado a Jackson 2.6+ a partir de la primavera 4.3)
- OkHttp 3.x (sigue siendo compatible con OkHttp 2.x de lado a lado)
- Netty 4.1
- Undertow 1.4
- Tomcat 8.5.2 y 9.0 M6
- Además, Spring 4.3 incorpora la versión actualizada ASM 5.1 y Objenesis 2.4 en spring-core.jar.

# La clase InjectionPoint

La clase **InjectionPoint** es una nueva clase introducida en Spring 4.3 que proporciona información sobre los lugares donde se inyecta un bean en particular, ya sea un parámetro método / constructor o un campo.

Los tipos de información que puedes encontrar usando esta clase son:

- **Objeto de campo**: puede obtener el punto de inyección envuelto como un objeto de campo usando el método getField () si el bean se inyecta en un campo
- **MethodParameter**: puede llamar al método getMethodParameter () para obtener el punto de inyección envuelto como un objeto MethodParameter si el bean se inyecta en un parámetro
- **Miembro**: el método getMember () devolverá la entidad que contiene el bean inyectado envuelto en un objeto Miembro
- **Clase <?>**: Obtenga el tipo declarado del parámetro o campo donde se inyectó el bean, usando getDeclaredType ()
- **Anotación []**: al usar el método getAnnotations (), puede recuperar una matriz de objetos de anotación que representan las anotaciones asociadas con el campo o parámetro
- **AnnotatedElement**: llame a getAnnotatedElement () para obtener el punto de inyección envuelto como un objeto AnnotatedElement

# La clase InjectionPoint

Un caso en el que esta clase es muy útil es cuando queremos crear beans de Logger basados en la clase a la que pertenecen:

```
@Bean
@Scope("prototype")
public Logger logger(InjectionPoint injectionPoint) {
    return Logger.getLogger(
      injectionPoint.getMethodParameter().getContainingClass());
}
```

El bean debe definirse con un ámbito de prototipo para que se cree un registrador diferente para cada clase. Si crea un bean singleton e inyecta en varios lugares, Spring recuperará el primer punto de inyección encontrado.

Luego, podemos inyectar el bean:

```
@Autowired
private Logger logger;
```

# Spring 5.1

# Revisión del framework para JDK 8

Spring Framework 5.0 se ha revisado para utilizar las nuevas funciones introducidas en Java 8. Las principales son:

- Sobre la base de las mejoras de reflexión de Java 8, se puede acceder de manera eficiente a los parámetros de los métodos en Spring Framework 5.0.
- Las interfaces Core Spring ahora proporcionan declaraciones selectivas basadas en los métodos predeterminados de Java 8.
- Las anotaciones @Nullable y @NotNull marcarán explícitamente los argumentos que pueden ser nulos y devolverán los valores. Esto permite tratar valores nulos en tiempo de compilación en lugar de lanzar NullPointerExceptions en tiempo de ejecución.

En el frente de registro, Spring Framework 5.0 sale de la caja con el módulo de puente de registro común, llamado spring-jcl en lugar del estándar de registro común. Además, esta nueva versión detectará automáticamente Log4j 2.x, SLF4J, JUL (java.util.logging) sin ningún puente adicional.

La programación defensiva también obtiene un impulso con la abstracción de recursos que proporciona el indicador isFile para el método getFile.

# Actualizaciones del contenedor core

Spring Framework 5.0 ahora admite el índice de componente candidato como una alternativa al escaneo de classpath. Este soporte se agregó al acceso directo al paso de identificación de componente candidato en el escáner de ruta de clase.

Una tarea de compilación de aplicaciones puede definir su propio archivo META-INF / spring.components para el proyecto actual. En el momento de la compilación, se inspecciona el modelo de origen y se marcan las entidades JPA y los Componentes Spring.

La lectura de entidades del índice en lugar de explorar el classpath no tiene diferencias significativas para proyectos pequeños con menos de 200 clases. Sin embargo, tiene impactos significativos en grandes proyectos. Cargar el índice de componentes es barato. Por lo tanto, el tiempo de inicio con el índice permanece constante a medida que aumenta el número de clases.

Lo que esto significa para nosotros los desarrolladores es que en los grandes proyectos de Spring, el tiempo de inicio de nuestras aplicaciones se reducirá significativamente. Mientras que 20 o 30 segundos no parecen mucho, cuando esperas esa cantidad de tiempo docenas o cientos de veces al día, se suma. Usar el índice de componentes te ayudará con tu productividad diaria.

# Actualizaciones del contenedor core

Ahora, las anotaciones @Nullable también pueden usarse como indicadores para puntos de inyección opcionales. El uso de @Nullable impone a los consumidores la obligación de que deben prepararse para que un valor sea nulo. Antes de esta versión, la única forma de lograr esto es a través de Nullable de Android, Nullable de Checker Framework y Nullable de JSR 305.

Algunas otras características nuevas y mejoradas de la nota de lanzamiento son:

- Implementación del estilo de programación funcional en GenericApplicationContext y AnnotationConfigApplicationContext.
- Detección consistente de transacciones, almacenamiento en caché y anotaciones asíncronas en los métodos de interfaz.
- Los espacios de nombres de configuración XML se simplifican hacia esquemas no versionados.

# Programación funcional con Kotlin

Spring Framework 5.0 introduce soporte para el lenguaje JetBrains Kotlin. Kotlin es un lenguaje orientado a objetos que admite el estilo de programación funcional. Kotlin se ejecuta sobre la JVM, pero no se limita a ella.

Con el soporte de Kotlin, los desarrolladores pueden sumergirse en la programación funcional de Spring, en particular, para los puntos finales funcionales web y el registro de beans.

En Spring Framework 5.0, puede escribir código de Kotlin limpio e idiomático para la API funcional de la Web de esta manera:

```
{
("/movie" and accept(TEXT_HTML)).nest {
GET("/", movieHandler::findAllView)
GET("/{card}", movieHandler::findOneView)
}
("/api/movie" and accept(APPLICATION_JSON)).nest {
GET("/", movieApiHandler::findAll)
GET("/{id}", movieApiHandler::findOne)
}
}
```

# Programación funcional con Kotlin

Para el registro de beans, como alternativa a XML o @Configuration y @Bean, ahora puede usar Kotlin para registrar sus Spring Beans de la siguiente manera:

```
val context = GenericApplicationContext {
registerBean()
registerBean { Cinema(it.getBean()) }
}
```

# Modelo de programación reactiva web

Una característica interesante es el nuevo marco web de pila reactiva. Al ser totalmente reactivo y sin bloqueo, esta pila es adecuada para el procesamiento de estilo de bucle de eventos que puede escalar con un pequeño número de subprocesos.

Reactive Streams es una especificación de API desarrollada por ingenieros de Netflix, Pivotal, Typesafe, Red Hat, Oracle, Twitter y Spray.io. Esto proporciona una API común para implementaciones de programación reactiva, como JPA para Hibernate, donde JPA es la API e Hibernate es la implementación.

La API de Reactive Streams es oficialmente parte de Java 9. En Java 8, deberá incluir una dependencia para la especificación de la API de Reactive Streams.

El soporte de transmisión en Spring Framework 5.0 se basa en Project Reactor, que implementa la especificación de la API de Reactive Streams.

# Modelo de programación reactiva web

Spring Framework 5.0 tiene un nuevo módulo spring-webflux que admite clientes HTTP y WebSocket reactivos. Spring Framework 5.0 también proporciona soporte para aplicaciones web reactivas que se ejecutan en servidores que incluyen REST, HTML e interacciones de estilo WebSocket.

Hay dos modelos de programación distintos en el lado del servidor en spring-webflux:

- Basado en anotaciones con @Controller y las otras anotaciones de Spring MVC.
- Estilo funcional de enrutamiento y manejo con Java 8 lambda.

Una implementación de WebClient de un punto final REST en Spring 5.0 es la siguiente:

```
WebClient webClient = WebClient.create();
Mono person = webClient.get()
.uri("http://localhost:8080/movie/42")
.accept(MediaType.APPLICATION_JSON)
.exchange()
.then(response -> response.bodyToMono(Movie.class));
```

# Mejoras para Pruebas

Spring Framework 5.0 es totalmente compatible con JUnit 5 Jupiter para escribir pruebas y extensiones en JUnit 5. Además de proporcionar un modelo de programación y extensión, el subproyecto de Jupiter proporciona un motor de prueba para ejecutar pruebas basadas en Jupiter en Spring.

Además, Spring Framework 5 proporciona soporte para la ejecución de pruebas paralelas en Spring TestContext Framework.

Para el modelo de programación reactiva, la prueba de primavera ahora incluye WebTestClient para integrar el soporte de pruebas para Spring WebFlux. El nuevo WebTestClient, similar a MockMvc, no necesita un servidor en ejecución. Usando una solicitud y respuesta simulada, WebTestClient puede unirse directamente a la infraestructura del servidor WebFlux.

# Mejoras para Pruebas

Para obtener una lista completa de mejoras en el marco existente de TestContext.

Por supuesto, Spring Framework 5.0 también es compatible con nuestro viejo amigo JUnit 4. El soporte para JUnit 4 será con el Framework Spring durante algún tiempo en el futuro.

# Librerías

Spring Framework 5.0 ahora admite las siguientes versiones actualizadas de la biblioteca:

- Jackson 2.6+
- EhCache 2.10+ / 3.0 GA
- Hibernate 5.0+
- JDBC 4.0+
- XmlUnit 2.x +
- OkHttp 3.x +
- Netty 4.1+

# Apoyo descontinuado a paquetes y librerías

En el nivel de API, Spring Framework 5.0 ha suspendido el soporte para los siguientes paquetes:

- *beans.factory.access*
- *jdbc.support.nativejdbc*
- *mock.static* del módulo de aspectos de spring.
- *web.view.tiles2M*. Ahora Tiles 3 es el requisito mínimo.
- *Orm.hibernate3 y orm.hibernate4*. Ahora, Hibernate 5 es el framework soportado.

Spring Framework 5.0 también ha suspendido el soporte para las siguientes bibliotecas:

- *Portlet*
- *Velocity.*
- *JasperReports.*
- *XMLBeans.*
- *JDO.*
- *Guayaba.*

Si está utilizando alguno de los paquetes anteriores, se recomienda permanecer en Spring Framework 4.3.x.

¿Preguntas?

# Fuentes

- http://www.java.sun.com
- https://spring.io/