



**VIEWNEXT**  
AN IBM SUBSIDIARY

**FORMACIÓN EN NUEVAS TECNOLOGÍAS**  
**REACTOR + Spring Integration + Spring WebFlux**

# ICONO TRAINING



# FORMADOR



## Formación en Nuevas Tecnologías



[www.iconotc.com](http://www.iconotc.com)



[linkedin.com/company/icono-training-consulting](https://linkedin.com/company/icono-training-consulting)



[training@iconotc.com](mailto:training@iconotc.com)

¡Síguenos en las Redes Sociales!



Ana Isabel Vegas



Consultora / formadora en Tecnologías  
de la Información.

# REACTOR + Spring Integration + WebFlux



## DURACIÓN

- ↳ 25 horas



## MODALIDAD

- ↳ Remoto.



## FECHAS y HORARIO:

- ↳ Días 15,16, 20, 21 y 22 de Enero de 2020 15:00-20:00.



## CONTENIDO:

- ↳ Introducción a los WebFlux
- ↳ Spring Boot
- ↳ Acceso a Datos
- ↳ WebClient
- ↳ Spring Integration
- ↳ Testing

# REACTOR + Spring Integration + WebFlux



## Introducción a los Webflux

↳ Conceptos

↳ Arquitectura

↳ Reactor



## Spring Boot

↳ Maven

↳ Starters

↳ Aplicaciones

↳ MVC

↳ Rest

↳ Serialización y Deserialización

↳ Programación Reactiva



## Acceso a Datos

↳ JPA

↳ H2

↳RestController

↳ Lambdas

↳ Mono y Flux

↳ MongoDB

↳ Consultas a Mongo DB

↳ Repositorio Mongo DB



## WebClient

↳ Llamadas HTTP

↳ Request Body

↳ Filtros



## Spring Integration

↳ Introducción

↳ Canales

↳ Cliente



## Testing

↳ Test de Unidad

↳ Test de Integración

↳ Test de Aceptación

↳ Test de Carga

↳ Reporting

# Introducción a los WebFlux

Tema 1

# WebFlux

## Spring WebFlux

¿Qué es Spring WebFlux?

↓

Es un marco de trabajo web reactivo que reemplaza el tradicional Spring MVC

¿Qué tipo de proyectos se puede hacer?

Diseñar arquitectura de microservicios (Spring Cloud WebFlux)

Aplicaciones web con Thymeleaf

Aplicaciones web con base de datos de drivers reactivos como MongoDB o implementaciones R2DBC

Servicios REST

Construcción de aplicaciones sin procesos bloqueantes

Utiliza programación reactiva

Utiliza paradigma funcional

Mejor rendimiento, menor consumo en memoria y aplicaciones escalables

Usos comunes para aplicaciones web

Crear un back-end reactivo de servicios rest e integrarlos con front-end como Angular, React.js, Vue.js

Crear monolito back-end reactivo de controladores y front-end thymeleaf

Thymeleaf

# Que es Spring WebFlux?

-  Desde la versión 5.x de Spring se introduce Spring Webflux como marco de trabajo (framework) de aplicaciones web reactivas.
-  Esta basado en reactive streams (<http://www.reactive-streams.org/>) y corre en servidores web como: Netty, Undertow, u contenedores que soporte la especificacion de Servlets 3.1 o superior.
-  Spring Webflux usa el proyecto **Reactor** como base para la implementación de reactive stream

# Arquitectura del Framework



**@Controller, @RequestMapping**

**Router Functions**

spring-webmvc

spring-webflux

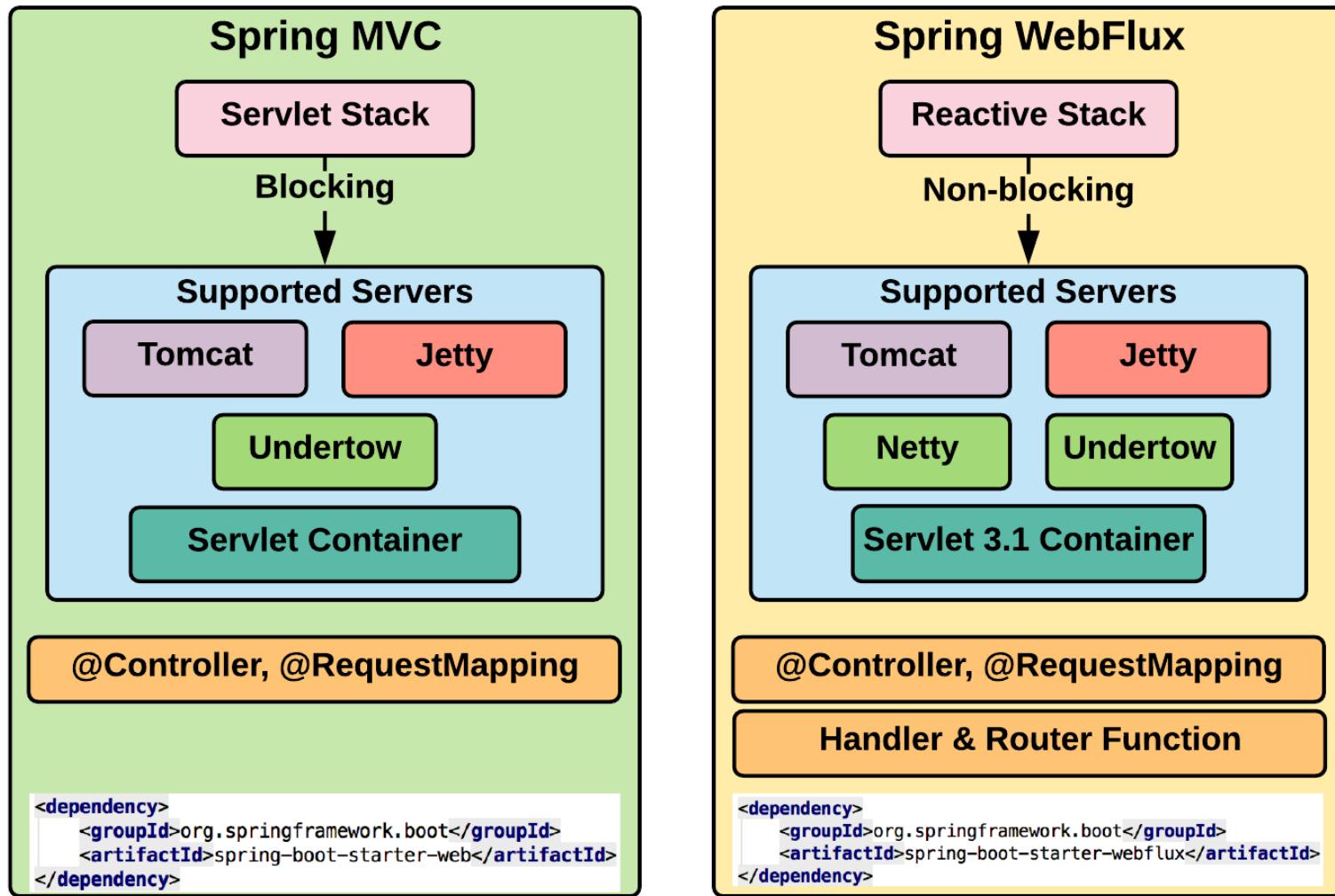
Servlet API

HTTP / Reactive Streams

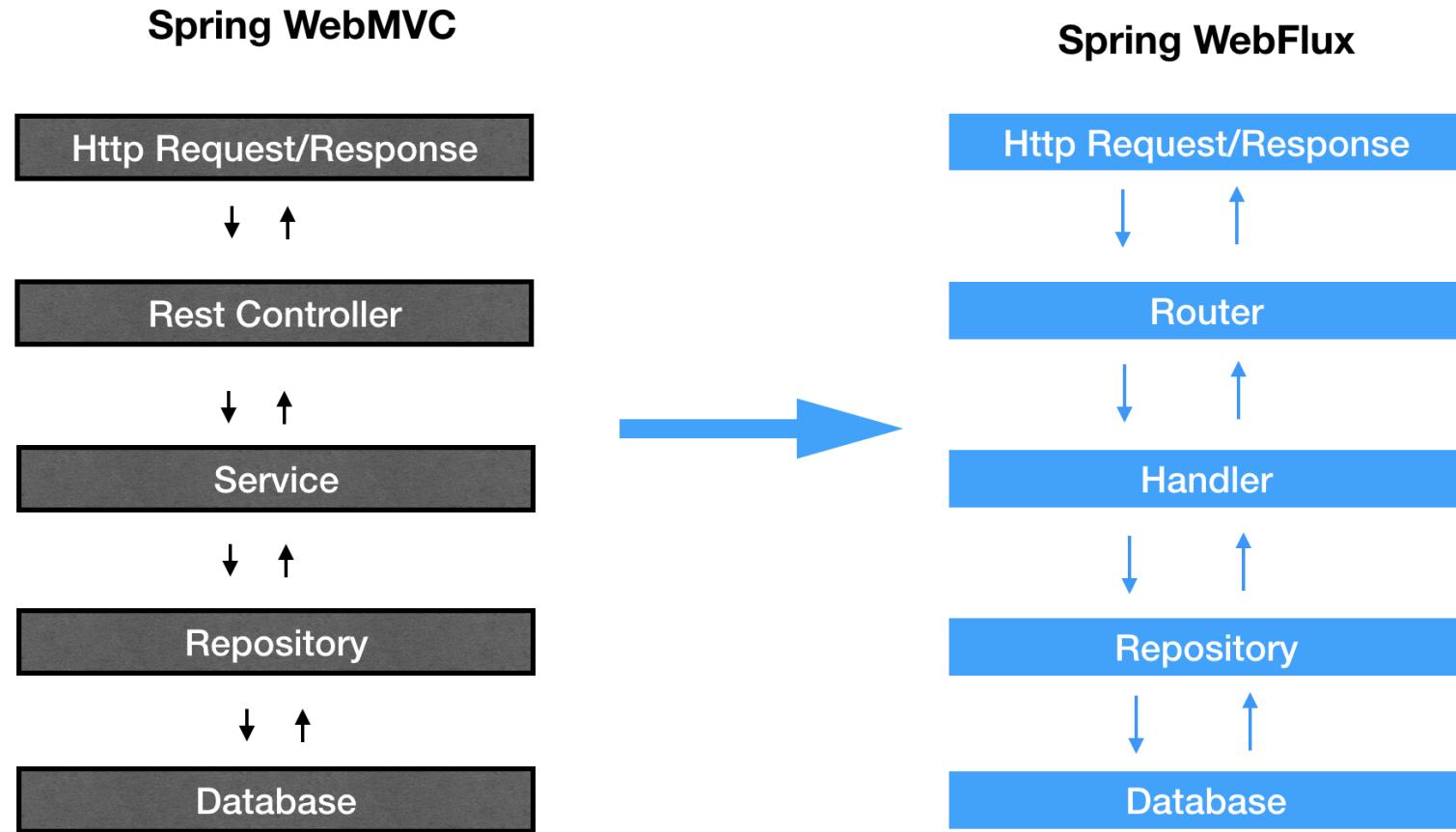
Servlet Container

Tomcat, Jetty, Netty, Undertow

# Spring MVC vs Spring WebFlux



# Componentes Spring WebFlux



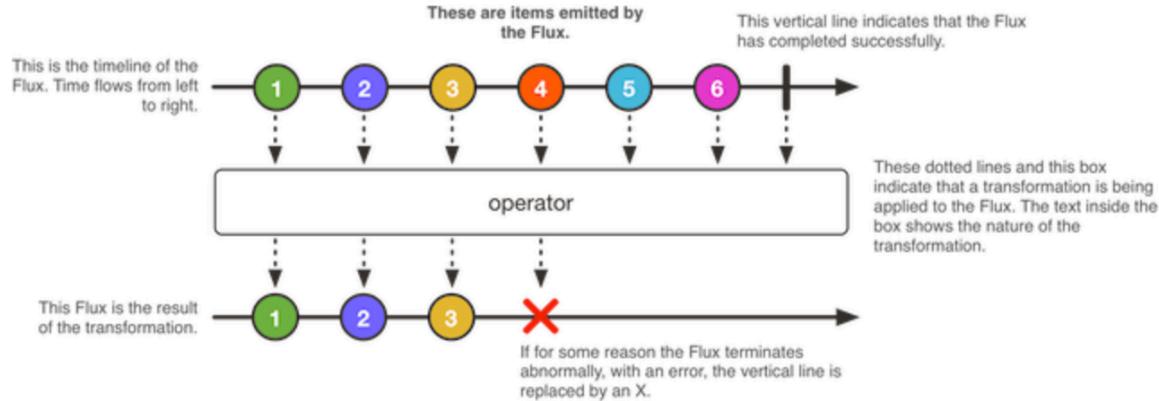
# Que es Reactor?

-  Fue concebido con la implicación del equipo responsable de RxJava 2 por lo que comparten gran parte de la base arquitectónica. Su principal ventaja es que al ser parte de Pivotal ha sido la elegida como fundación del futuro **Spring 5 WebFlux Framework**.
-  Este API introduce los tipos Flux y Mono como implementaciones de Publisher, los cuales generan series de 0...N y 0...1 elementos respectivamente.
-  **Flux:** Flux hace referencia a un conjunto de elementos .
-  **Mono:** Mono hace referencia a un elemento de programación asincrona a uno solo

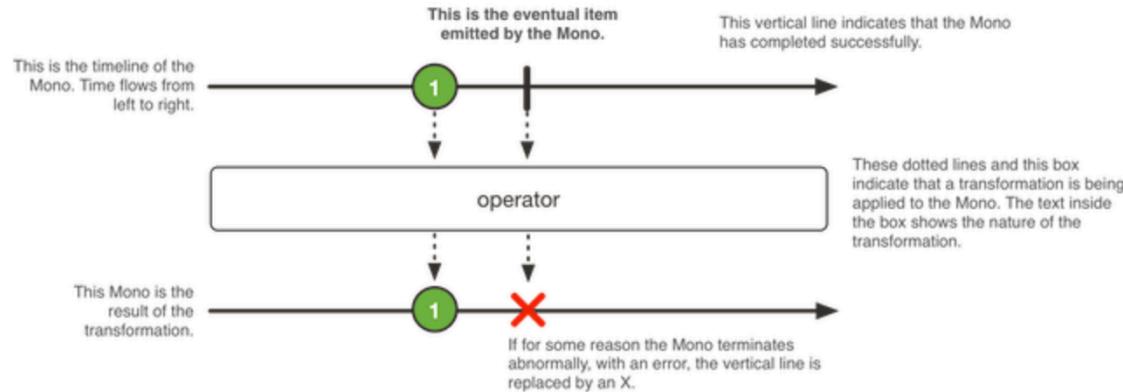
# Flux vs Mono



Flux:



Mono:



# Spring Boot

Tema 2

# Que es Spring Boot

-  Spring Boot es una parte de Spring que nos permite crear diferentes tipos de aplicaciones de una manera rápida y sencilla.
-  Sus características principales son que provee out-of-the-box una serie de elementos que nos permiten desarrollar diferentes tipos de aplicaciones de forma casi inmediata. Algunas de estas características son:
  -  Servidores de aplicaciones embebidos (Tomcat, Jetty, Undertow)
  -  POMs con dependencias y plug-ins para Maven
  -  Uso extensivo de anotaciones que realizan funciones de configuración, inyección, etc.

# Configuración del pom

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
</parent>

<properties>
    <java.version>1.8</java.version>
</properties>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.1.11</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-logging</artifactId>
        <version>2.1.4.RELEASE</version>
    </dependency>
</dependencies>
```

# Principales Anotaciones

-  La etiqueta **@Configuration**, indica que la clase en la que se encuentra contiene la configuración principal del proyecto.
-  La anotación **@EnableAutoConfiguration** indica que se aplicará la configuración automática del starter que hemos utilizado. Solo debe añadirse en un sitio, y es muy frecuente situarla en la clase main.
-  En tercer lugar, la etiqueta **@ComponentScan**, ayuda a localizar elementos etiquetados con otras anotaciones cuando sean necesarios.
-  Para no llenar nuestra clase de anotaciones, podemos sustituir las etiquetas **@Configuration**, **@EnableAutoConfiguration** y **@ComponentScan** por **@SpringBootApplication**, que engloba al resto.

# Clase principal

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloWorldApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloWorldApplication.class, args);
    }
}
```

## 2. Spring Boot

- a. Maven
- b. Starters
- C. Aplicaciones
- d. MVC
- e. Rest
- f. Serialización y Deserialización
- g. Programación Reactiva

## 2.a. Acerca de MAVEN

- a. Que es MAVEN
- b. Objetivos de MAVEN
- C. Instalación
- d. Plugin para Eclipse

## 2.a.a. Que es MAVEN



Maven es una herramienta para la gestión y construcción de proyectos Java fue creada por Jason Van Zyl. De sonatype en 2002.



Sus principales características son:

- Funcionalidad similar a Apache Ant
- Configuración por xml
- Para cada proyecto tiene un fichero llamado pom.xml (Project Object Model) para describir el proyecto de software a construir, sus dependencias de otros módulos o componentes externos, etc.

## 2.a.b. Objetivos de MAVEN

-  Facilitar la compilación y paquetización.
  -  Provee herramientas que facilitan estos procesos.
-  Proveer una forma uniforme de construir sistemas.
  -  Utiliza una configuración llamada POM (Project Object Model), y una serie de plugins que pueden ser utilizados por los proyectos.
  -  Define una forma estándar de estructurar el código fuente, las dependencias, y las herramientas necesarias.
-  Proveer información de la calidad del proyecto.
  -  Se integra con herramientas que permiten evaluar la calidad del código, incluyendo dependencias, testing, y también codificación.
-  Promover buenas prácticas de desarrollo.
  -  Permite ejecutar tests, manteniendo su código fuente separado del principal.
-  Facilitar la migración a nuevas características.
  -  La actualización consiste simplemente en renovar las versiones de los plugins declarados en el POM, y aplicar los cambios.

## 2.a.c. Instalación

-  Antes de instalar maven tenemos que asegurarnos de tener:
  -  JDK instalado en nuestra máquina. Es muy importante que no sea tan solo JRE.
-  Una vez cumplidos los requisitos ya podemos proceder a la descarga desde la pagina de apache:
  -  <https://maven.apache.org/download.cgi>

<a href="https://maven.apache.org/download.cgi">https://maven.apache.org/download.cgi</a>			
against the public <b>KEYS</b> used by the Apache Maven developers.			
	Link	Checksum	Signature
Binary tar.gz archive	<a href="#">apache-maven-3.3.9-bin.tar.gz</a>	<a href="#">apache-maven-3.3.9-bin.tar.gz.md5</a>	<a href="#">apache-maven-3.3.9-bin.tar.gz.asc</a>
Binary zip archive	<a href="#">apache-maven-3.3.9-bin.zip</a>	<a href="#">apache-maven-3.3.9-bin.zip.md5</a>	<a href="#">apache-maven-3.3.9-bin.zip.asc</a>
Source tar.gz archive	<a href="#">apache-maven-3.3.9-src.tar.gz</a>	<a href="#">apache-maven-3.3.9-src.tar.gz.md5</a>	<a href="#">apache-maven-3.3.9-src.tar.gz.asc</a>
Source zip archive	<a href="#">apache-maven-3.3.9-src.zip</a>	<a href="#">apache-maven-3.3.9-src.zip.md5</a>	<a href="#">apache-maven-3.3.9-src.zip.asc</a>

## 2.a.c. Instalación



La herramienta Maven es un directorio que al descomprimir vemos que contiene las siguientes subcarpetas:

- **bin**: comandos, donde se encuentra el principal: mvn
- **boot**
- **conf**: contiene la configuración principal, el archivo “settings.xml”
- **lib**: las librerías de Maven

## 2.a.d. Plugin para Eclipse



A partir de la versión Luna ya incluye el plugin de Maven.



Para comprobar si existe se accede a Window -> Preferences -> Maven



Si queremos trabajar con una versión posterior, se agrega a través de la opción installation, referenciando al directorio que nos acabamos de descargar y descomprimir.



Además, es necesario especificar el settings.xml "de usuario", en el menú Maven → User Settings.

Esta configuración prevalece por sobre la global. Una opción, que es la aplicada en este curso, es utilizar el mismo settings.xml para ambos casos, global y de usuario.

### ▼ Maven

- Annotation Processing
- Archetypes
- Discovery
- Errors/Warnings
- Installations**
- Java EE Integration
- Lifecycle Mappings

Select the installation used to launch Maven:

Name	Details
<input type="checkbox"/> EMBEDDED	3.3.9/1.8.2.20171007-0216
<input type="checkbox"/> WORKSPACE	⚠ NOT AVAILABLE [3.0.]
<input checked="" type="checkbox"/> apache-maven-3.6.1	/Users/anaisabelvegascaceres/Documents/apache-maven-3.6.1 3.6.1

## 2.a. Estructura de proyectos

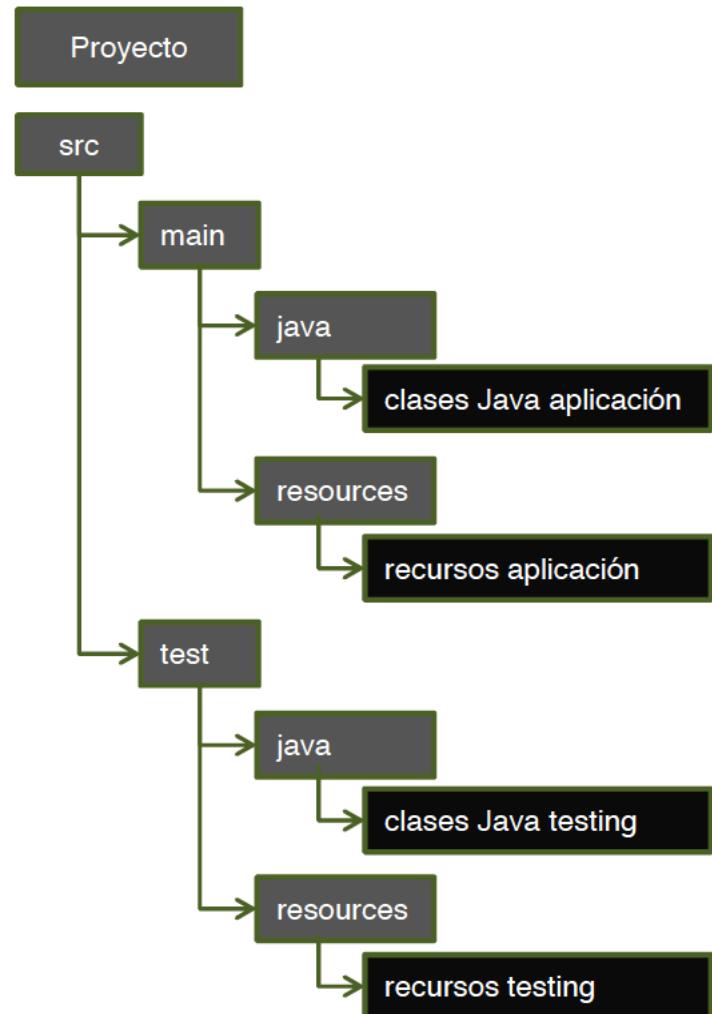


Maven especifica una ubicación estándar para el código fuente del proyecto, distinguiendo entre:

- Código Java de aplicación
- Recursos de aplicación
- Código Java de testing
- Recursos de testing.



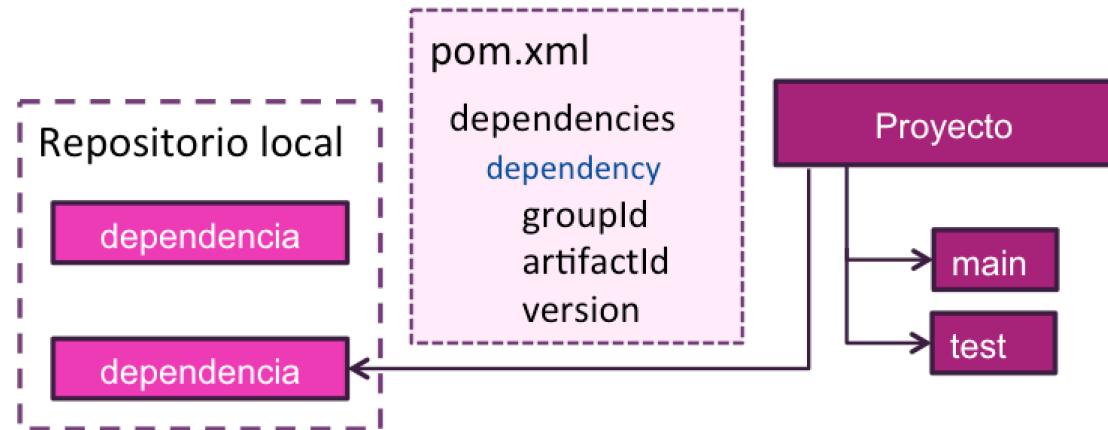
Esto, aparte de facilitar la distinción de tipo de código fuente, permite que cuando se paquetiza el proyecto, no se incluya el código de testing.



## 2.a. Gestión de dependencias

- Una de las principales características de Maven es el manejo de dependencias.
- Si se tiene un proyecto Java que depende de una librería, por ejemplo "hibernate-core", dicha librería no se almacena dentro del proyecto, sino en un repositorio local. Para ello, Maven la identifica con tres elementos en el tag <dependencies> del archivo POM, a las que se llama "**coordenadas**". Ejemplo:

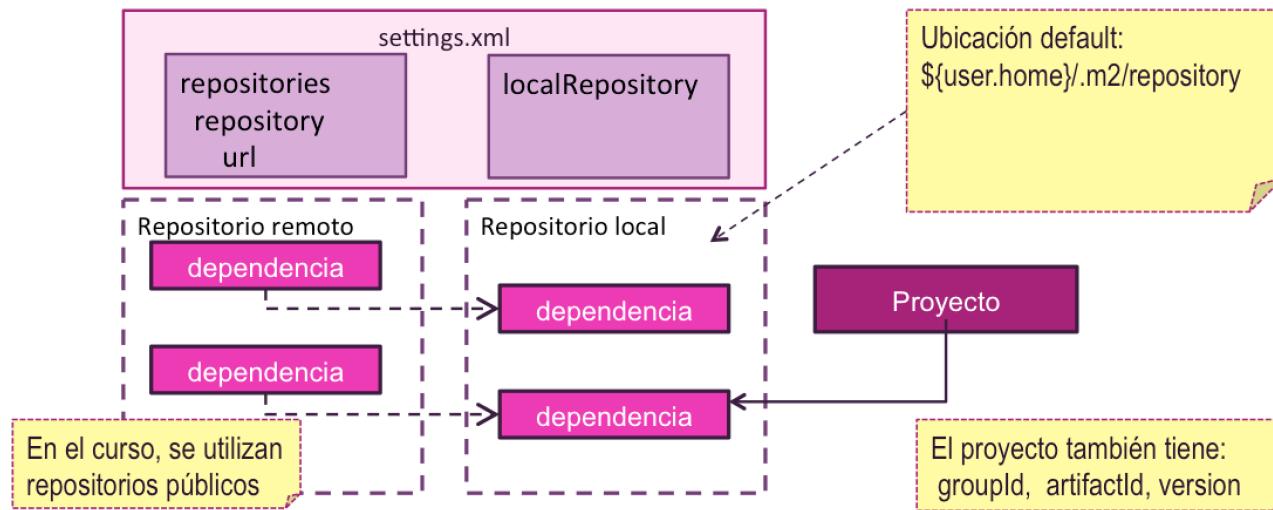
- groupId: hibernate
- artifactId: hibernate-core
- version: 4.1.8.Final



## 2.a. Gestión de dependencias



Dada una **dependencia** identificada por el **groupId**, **artifactId** y **version**, Maven verifica si existe en el repositorio local. Si no está, lo intenta buscar en los repositorios remotos configurados, ya sea en el propio POM, otro POM relacionado, o en el **settings.xml**, y lo copia al repositorio local:



## 2.a. Gestión de dependencias



Existen 6 ámbitos en los que una dependencia puede ser declarada limitando así su transitividad.

- ↳ **compile**: ámbito por defecto. Las dependencias están disponibles en el proyecto y en sus proyectos dependientes.
- ↳ **provided**: se espera que el JDK, la aplicación o el contenedor provea la dependencia. Por ejemplo, al crear una aplicación web Java Enterprise, se debe establecer la dependencia de las APIs Servlet y Java EE relacionadas con el alcance *provided* ya que el contenedor web proporciona esas clases. Este ámbito de aplicación sólo está disponible en la compilación y test, y no es transitiva
- ↳ **runtime**: la dependencia no es requerida en tiempo de compilación pero sí para la ejecución.
- ↳ **test**: son dependencias que son requeridas solo cuando se compila y ejecuta los test.
- ↳ **system**: similar a provided pero se le debe indicar el jar que contiene la dependencia
- ↳ **import**: Únicamente utilizado en dependencias de tipo pom en el apartado "" . De este modo se importan las dependencias definidas en otro pom

## 2.a. Gestión de dependencias

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
    <scope>compile</scope>
</dependency>

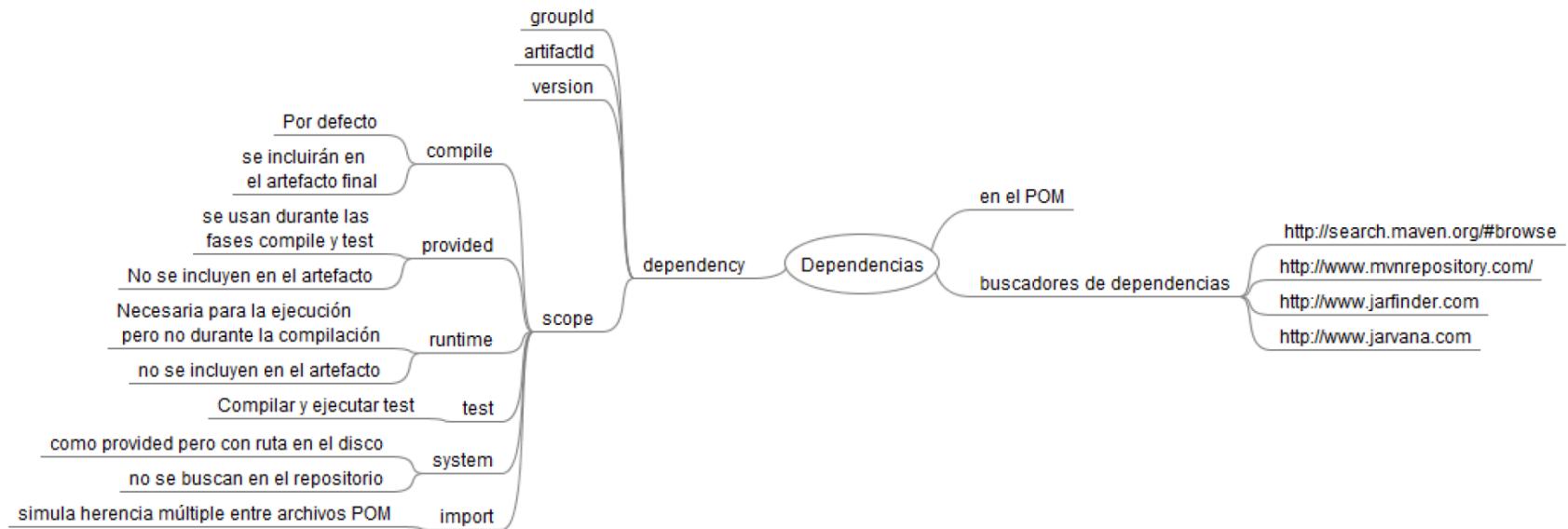
<dependency>
    <groupId>javax.el</groupId>
    <artifactId>el-api</artifactId>
    <version>1.0</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>sun.jdk</groupId>
    <artifactId>tools</artifactId>
    <version>1.5.0</version>
    <scope>system</scope>
    <systemPath>${java.home}/../lib/tools.jar</systemPath>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.12</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.2</version>
    <scope>test</scope>
</dependency>
```

## 2.a. Gestión de dependencias



## 2.a. Gestión de dependencias



Maven se puede configurar en tres niveles:



Proyecto



Instalación descargada



Usuario



El repositorio local por defecto esta en \${user.home}/.m2/repository.



Podemos modificar esta ubicación indicandola en el fichero settings.xml. Ojo!!! Debe ser una ruta absoluta

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.ap
  <!-- localRepository
    | The path to the local repository maven will use to store artifacts.
    |
    | Default: ${user.home}/.m2/repository
  -->
<localRepository>D:\proyectos\Maven\java-env\maven_repo</localRepository>
```

## 2.b. Starters

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

## 2.c. Aplicaciones



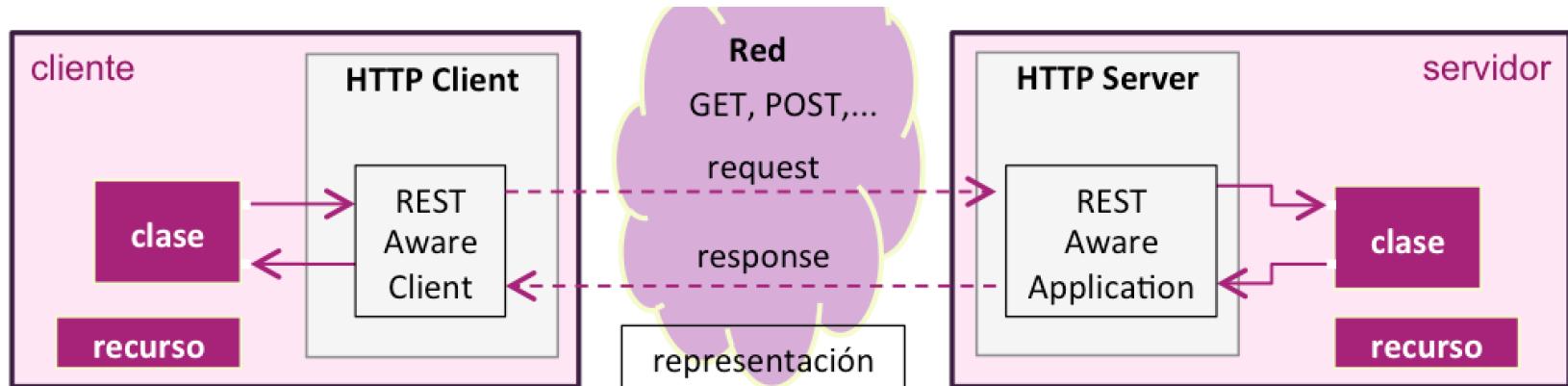
Con Spring Boot podemos crear aplicaciones tanto locales, como web, ...etc.

## 2.d. MVC

-  Spring Mvc es una alternativa de framework basado en el patrón modelo-vista-controlador. Es un modulo más dentro del framework Spring.
-  La lógica de negocio asociada a cada petición la resolverá un controlador asociado a esta. Estos controladores se denominan Controllers.
-  Podemos utilizar anotaciones para mapear el path con su controller correspondiente. Ahora sobre el propio controller utilizamos la anotación RequestMapping para establecer el path de la aplicación.

## 2.e. REST

-  **REST (Representational State Transfer)** es un estilo de arquitectura para sistemas distribuidos, desarrollada por la W3C, junto con el protocolo HTTP.
-  Las arquitecturas REST tienen clientes y servidores.
-  El cliente realiza un envío (request) al servidor, el cual lo procesa y retorna una respuesta al cliente.
-  Las peticiones y respuestas son construidas alrededor de representaciones de recursos. Recurso es una entidad, y representación es cómo se formatea.



## 2.e. Building a RESTful Web Service



Una API del tipo RESTful, o RESTful Web Service, es una API web implementada con HTTP y los principios REST, con los siguientes aspectos:

- ↳ Una URI base del servicio.
- ↳ Un formato de mensajes, por ejemplo JSON o XML.
- ↳ Un conjunto de operaciones, que utilizan los métodos HTTP (GET, PUT, POST o DELETE).



La API debe manejar hipertextos.



A diferencia de los Web Services basados en SOAP, no hay un estándar comúnmente aceptado para los RESTful. Esto es porque REST es una arquitectura, mientras que SOAP es un protocolo.



Esta desventaja se compensa con la simplicidad de su utilización y el bajo consumo de recursos durante el binding. Esto es especialmente útil en aplicaciones para dispositivos móviles

## 2.e. Building a RESTful Web Service



Con REST, los **métodos HTTP** se asocian a tipos de operaciones sobre recursos. El uso comúnmente aceptado es el siguiente:

- **GET**: Para recuperar la representación de un recurso. Es idempotente, es decir, si se invoca múltiples veces, retorna el mismo resultado.
- **POST**: Para crear un recurso, o para actualizarlo. También, por las características del método, se utiliza para envíos grandes, o para evitar limitaciones de los otros métodos.
- **PUT**: Para actualizar un recurso, ya que POST no es idempotente.
- **DELETE**: Para eliminar un recurso.
- **OPTIONS**: Se puede utilizar para hacer un "ping" del servicio, es decir, verificar su disponibilidad.
- **HEAD**: Para buscar un recurso o consultar estado. Similar a GET, pero no contiene un body.

## 2.e.Building a RESTful Web Service

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SaludoRest {

    // http://localhost:8080/hola
    @RequestMapping("/hola")
    public String hola() {
        return "Bienvenidos al curso";
    }

    // http://localhost:8080/adios?usuario="Anabel"
    @RequestMapping("/adios")
    public String adios(@RequestParam(value="usuario", defaultValue="Admin") String user) {
        return "Nos vamos a desayunar " + user;
    }
}
```

## 2.e. Building a RESTful Web Service



Al agregar esta dependencia al pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```



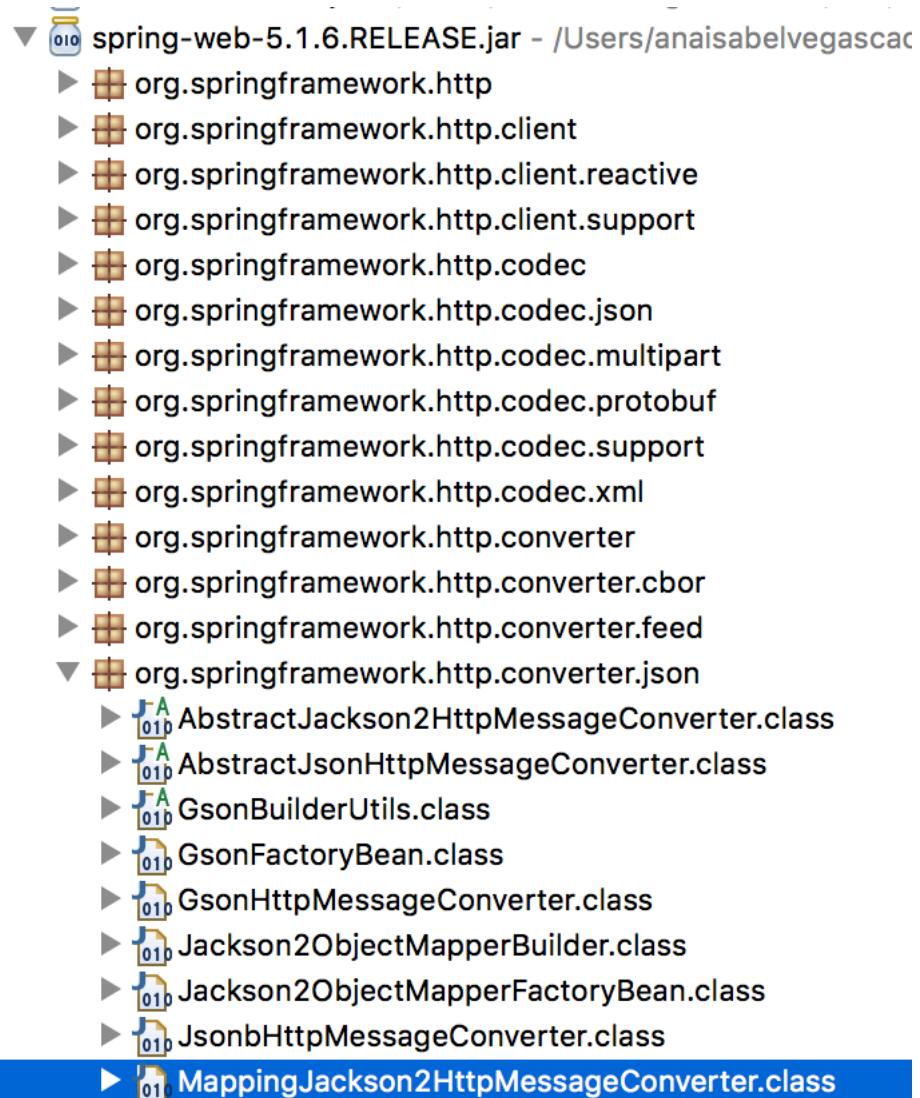
La clase `MappingJackson2HttpMessageConverter` se encarga de convertir automáticamente la instancia a devolver en un formato JSON.

## 2.e. Building a RESTful Web Service



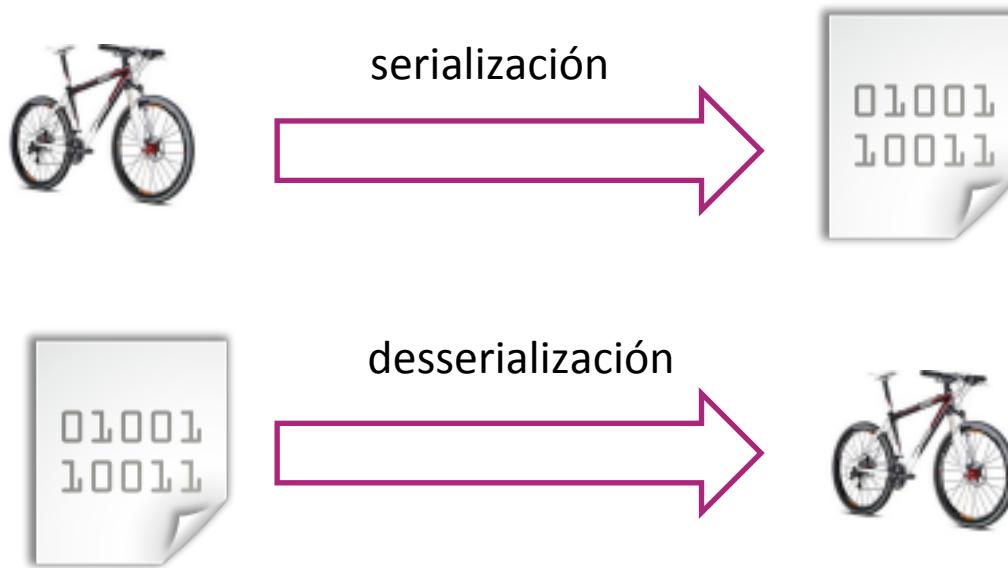
Formateando la respuesta a formato JSON:

### Dependency Hierarchy



## 2.f. Serialización y Deserialización

-  La **serialización** es un caso de entrada/salida, en la cual un objeto es convertido en una cadena de bytes para ser enviado a un destino.
-  La **deserialización** es el proceso inverso, es decir, desde una cadena de bytes se puede recuperar el objeto.



## 2.f. Serialización y Deserialización



## ejecución de serialización



Para **serializar** un objeto, se utiliza la clase `ObjectOutputStream`, la cual recibe como parámetro otro `OutputStream` con el destino donde se escribe el objeto serializado.

```
Calendar cal = Calendar.getInstance();
cal.set(2013, Calendar.JULY, 14, 0, 0);

File file = new File("files", "cal.bin");
FileOutputStream fos =
        new FileOutputStream(file);
ObjectOutputStream oos =
        new ObjectOutputStream(fos);
oos.writeObject(cal); <-----  
fos.close();
```

Crea objeto Calendar con fecha y hora específica.

Serializa el objeto Calendar, escribiendo sobre el FileOutputStream el resultado.

```
000000000h: AC ED 00 05 73 72 00 1B 6A 61 76 61 2E 75 74 69 ; -i..sr..java.util  
00000010h: 6C 2E 47 72 65 67 6F 72 69 61 6E 43 61 6C 65 6E ; GregorianCalen  
00000020h: 64 61 72 8F 3D D7 D6 E5 B0 D0 C1 02 00 01 4A 00 ; dar=>ÖDÁ...J.  
00000030h: 10 67 72 65 67 6F 72 69 61 6E 43 75 74 6F 76 65 ; .gregorianCutove  
00000040h: 72 78 72 00 12 6A 61 76 61 2E 75 74 69 6C 2E 43 ; rxr..java.util.C  
00000050h: 61 6C 65 6E 64 61 72 E6 EA 4D 1E C8 DC 5B 8E 03 ; alendar=ëM.ËÜ[Z.  
00000060h: 00 0B 5A 00 0C 61 72 65 46 69 4C 7C 64 73 53 65 ; ..? `reFieldsSe  
00000070h: 74 40 79 72 77 4F 66 57 -tDayOfWe  
00000080h: -eSetZ.
```

Resultado: archivo  
cal.bin, 2.716 bytes

## 2.f. Serialización y Deserialización



## ejecución de deserialización



Para **deserializar** una cadena de bytes hacia un objeto, se utiliza la clase `ObjectInputStream`, la cual recibe como parámetro otro `InputStream` con el origen desde donde se lee el objeto serializado.

```
00000000h: AC ED 00 05 73 72 00 1B 6A 61 76 61 2E 75 74 69 ; -i..sr..java.util  
00000010h: 6C 2E 47 72 65 67 6F 72 69 61 6E 43 61 6C 65 6E ; 1.GregorianCalen  
00000020h: 64 61 72 8F 3D D7 D6 E5 B0 D0 C1 02 00 01 4A 00 ; dar=>ÖÄ°Đ...J.  
00000030h: 10 67 72 65 67 6F 72 69 61 6E 43 75 74 6F 76 65 ; .gregorianCutove  
00000040h: 72 78 72 00 12 6A 61 76 61 2E 75 74 69 6C 2E 43 ; rxr..java.util.C  
00000050h: 61 6C 65 6E 64 61 72 E6 EA 4D 1E C8 DC 5B 8E 03 ; alendar&M.ÈU[Ž.  
00000060h: 00 0B 5A 00 0C 61 72 65 46 69 Fc °C 64 73 53 65 ; ..? `reFieldsSe  
00000070h: 74 40 ^ 79 72 ?? 4F 66 ?? - `tDayOfWe  
00000080h: eSetZ. `inim `se
```



```
File file = new File("files", "cal.bin");
FileInputStream fis =
        new FileInputStream(file);
ObjectInputStream ois =
        new ObjectInputStream(fis); ↗
Calendar cal = (Calendar) ois.readObject();
fis.close();
```

Lee objeto serializado desde el FileInputStream.

## Deserializa Calendar

```
System.out.println(cal.getTime()); ➔ Sun Jul 14 00:00:00 CEST 2013
```

## 2.f. Serialización y Deserialización



condiciones de serializable



Para que un objeto se pueda serializar, debe cumplir con una de las dos siguientes condiciones:

- Ser marcado con la interfaz **Serializable**. Es la opción comúnmente utilizada. Serializa el objeto y sus atributos **de instancia**. Opcionalmente, se pueden implementar los métodos **writeObject** y **readObject** para modificar el comportamiento de la serialización y deserialización.
  - Implementar la interfaz **Externalizable**. Permite construir, a través de los métodos **writeExternal** y **readExternal**, una serialización y deserialización personalizadas. Tiene mayor flexibilidad que la forma default de Serializable, para los casos especializados, aunque en muchos casos no es necesaria.
- El objeto **ObjectOutputStream** internamente verifica que el objeto cumpla con la interfaz Externalizable, y si no, Serializable. Si no cumple con ninguna, lanza una **NotSerializableException**.

## 2.f. Serialización y Deserialización



### condiciones de serializable



Para que se pueda ejecutar la serialización de una clase **Serializable**, se deben dar las siguientes condiciones:



Si **no** se implementa el método `writeObject`:

- Todos sus atributos de instancia (no static) deben ser serializables.
- Si algún atributo no es serializable, puede ser marcado con el modificador `transient`.  
En ese caso, no se envía, recuperándose en la deserialización como `null`.
- También se pueden marcar como `transient` atributos serializables, en caso que no se quieran incluir en la serialización.
- Si se implementa el método `writeObject`, y no se utiliza la invocación a `defaultWriteObject()`, pueden haber atributos no serializables, siempre que no se utilicen.

## 2.f. Serialización y Deserialización



serialización default



Ejemplo de clase Serializable, con atributos serializables, que representa una fecha sin hora:

```
public class SerialDate implements Serializable {  
  
    private int date;  
    private int month; <----- Atributos serializables,  
    private int year;      por ser primitivos.  
  
    public SerialDate(int date, int month, int year) {  
        this.date = date;  
        this.month = month;  
        this.year = year;  
    }  
  
    // getters
```

**new SerialDate(14, 3, 2012)**



A hex dump of the serialized bytes for the date, month, and year fields. The bytes are arranged in a grid:

AC	ED	00	05	73	72	00	19	63	6F	6D	2E	62	70	65	2E
73	65	72	69	61	6C	2E	53	65	72	69	61	6C	44	61	74
65	00	00	00	00	00	00	00	01	02	00	03	49	00	04	64
61	74	65	49	00	05	6D	6F	6E	74	68	49	00	04	79	65
61	72	78	70	00	00	00	0E	00	00	00	03	00	00	07	DC

The diagram shows the mapping of the serialized bytes back to their corresponding primitive integer values:

int date 14	int month 3	int year 2012
----------------	----------------	------------------

## 2.f. Serialización y Deserialización



### atributos no serializables



Si un atributo de la clase no es (ni puede ser) serializable, aunque la clase lo sea, al intentar serializar lanza un NotSerializableException. Para evitarlo, el atributo se puede colocar como transient, siendo ignorado y llegando null.

```
public class SerialDate implements Serializable {  
  
    private int date;  
    private int month;  
    private int year;  
  
    private transient URLConnection connection;  
  
    public SerialDate(int date, int month, int year) {  
        this.date = date;  
        this.month = month;  
        this.year = year;  
    }  
  
    // getters
```

transient: no se incluye  
en la serialización.

## 2.f. Serialización y Deserialización



atributos no serializables



También se podría tener un atributo que interesa que no se serialice:

```
public class SerialDate implements Serializable {  
  
    private int date;  
    private int month;  
    private int year;  
  
    private transient String password;  
    private transient String hugeData;  
  
    public SerialDate(int date, int month, int year) {  
        this.date = date;  
        this.month = month;  
        this.year = year;  
    }  
  
    // getters
```

Aunque sea serializable, puede interesar que **no** lo sea por tamaño, confidencialidad u otra razón.

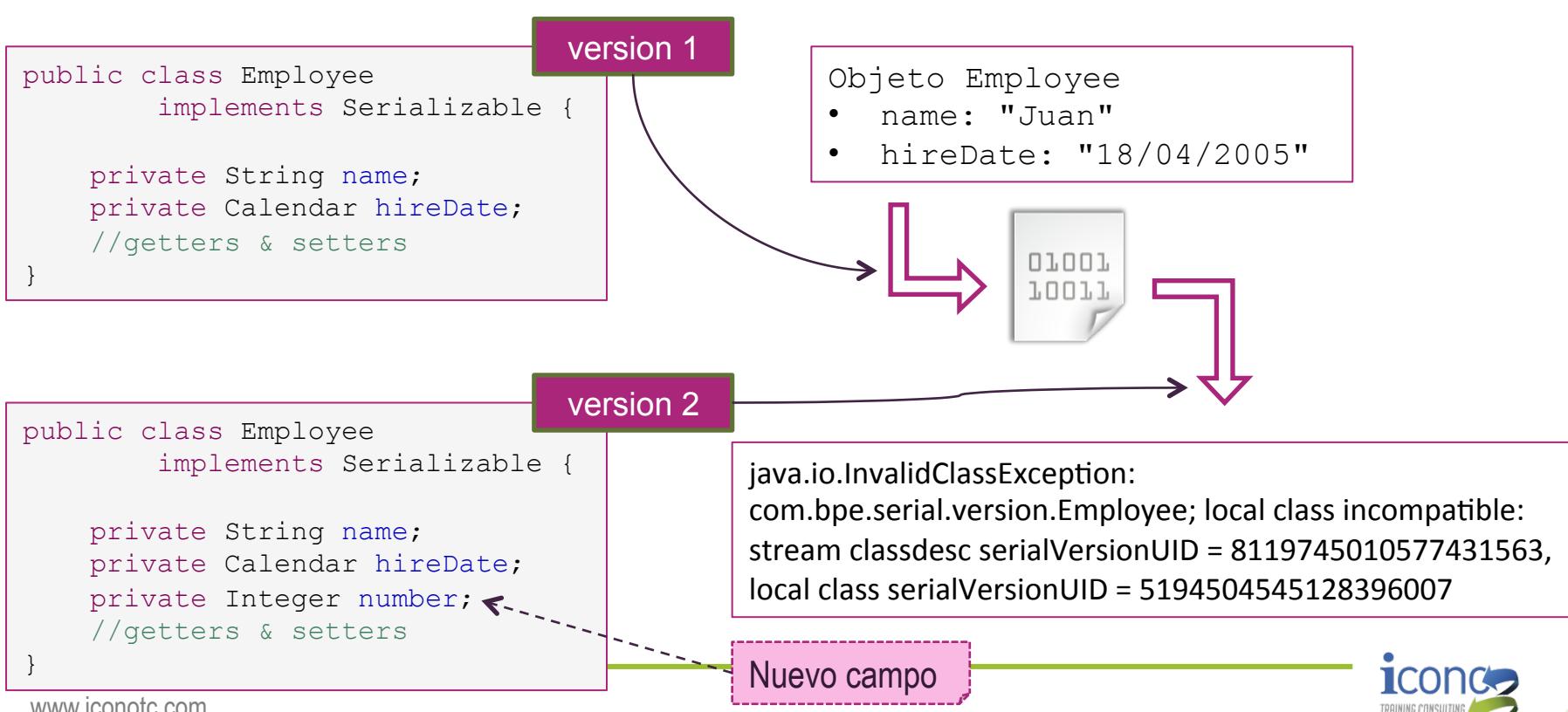
## 2.f. Serialización y Deserialización



versión de clase



Cuando un objeto se serializa, y luego cambia la versión de la clase, podría haber un problema al deserializar, ya que no se reconoce la clase original.



## 2.f. Serialización y Deserialización



### versión de clase



En caso que la nueva clase sea compatible con la original, es decir, tiene el mismo paquete, nombre y atributos, la forma de manejar la compatibilidad es a través de la utilización de un atributo long llamado **serialVersionUID**:

```
public class Employee  
    implements Serializable {  
    >private static final  
        long serialVersionUID = 1L;  
    private String name;  
    private Calendar hireDate;  
    //getters & setters  
}
```

version 1

Objeto Employee  
• name: "Juan"  
• hireDate: "18/04/2005"

```
public class Employee  
    implements Serializable {  
    >private static final  
        long serialVersionUID = 1L;  
    private String name;  
    private Calendar hireDate;  
    private Integer number;  
    //getters & setters  
}
```

version 2

Objeto Employee  
• name: "Juan"  
• hireDate: "18/04/2005"  
• number: **null**

Mismo valor

En la deserialización, si hay diferencias compatibles, y coinciden los serialVersionUID, entonces se ejecuta correctamente.

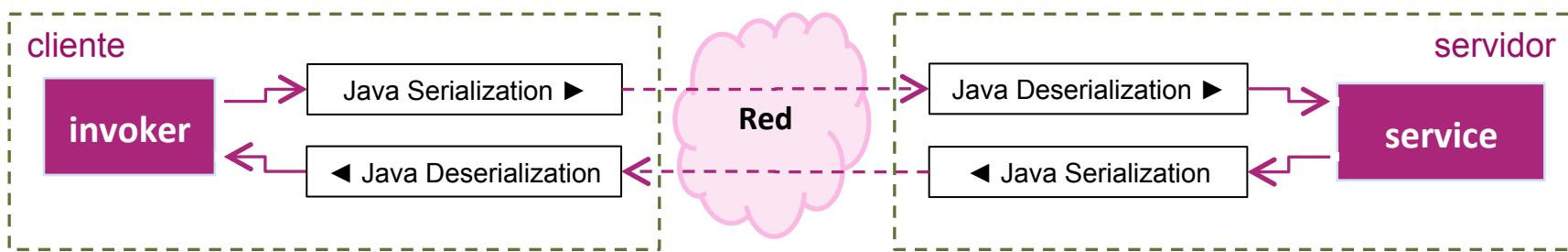
## 2.f. Serialización y Deserialización



aplicación



Aunque los objetos se pueden almacenar serializados, la mayor utilidad de la serialización es la invocación remota en un entorno cliente/servidor:



En la invocación (lado **cliente**), los parámetros del método son serializados, enviados a través de la red, y deserializados (lado **servidor**). Después de la ejecución del método, el objeto de retorno es serializado por el servidor, enviado a través de la red, y deserializado por el lado cliente.



**Nota:** la invocación remota no se implementa manualmente. Para ello existen soluciones que se describen en los siguientes cursos. Esto es sólo ilustrativo.

## 2.g. Programación Reactiva

- 💡 La **programación reactiva** es un paradigma enfocado en el trabajo con flujos de datos finitos o infinitos 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.

# **Acceso a Datos**

Tema 3

### 3. Acceso a Datos

- a. JPA
- b. H2
- C. RestController
- d. Lambdas
- e. Mono y Flux
- f. MongoDB
- g. Consultas a MongoDB
- h. Repositorio MongoDB

## 3.a. JPA

-  JPA es el acrónimo de **Java Persistence API** y se podría considerar como el estándar de los frameworks de persistencia.
-  En JPA utilizamos anotaciones como medio de configuración.
-  Consideramos una **entidad** al objeto que vamos a persistir o recuperar de una base de datos. Se puede ver una entidad como la representación de un registro de la tabla.
-  Toda entidad ha de cumplir con los siguientes requisitos:
  - Debe implementar la interface Serializable
  - Ha de tener un constructor sin argumentos y este ha de ser público.
  - Todas las propiedades deben tener sus métodos de acceso get() y set().
-  Para crear una entidad utilizamos la anotación **@Entity**, con ella marcamos un POJO como entidad.

## 3.b. H2



Vamos a trabajar con una base de datos en memoria H2, para ello necesitamos agregar la siguiente dependencia al pom.xml

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

# Accessing JPA Data with REST



Spring nos facilita el trabajar con los datos incluyendo estas dependencias:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

# Accessing JPA Data with REST



Debemos mapear la entidad a manejar en la base de datos:

```
@Entity  
public class Producto {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private long id;  
  
    private String descripcion;  
    private double precio;
```

# Accessing JPA Data with REST



Y por ultimo tener el repositorio donde se generaran las queries de forma automatica;

```
@RepositoryRestResource(collectionResourceRel = "productos", path = "productos")
public interface ProductoRepository extends PagingAndSortingRepository<Producto, Long> {

    List<Producto> findByDescripcion(@Param("descripcion") String descripcion);

}
```

# Accessing JPA Data with REST



Una vez levantada la aplicación con Spring Boot ya podemos probarla con los siguientes comandos en consola:

```
// comandos a ejecutar en consola "con permiso de administrador"
// Acceso al servicio
// curl http://localhost:8080

// Consultar todos los productos
// curl http://localhost:8080/productos

// Alta de producto y consulta para su verificacion
// curl -i -X POST -H "Content-Type:application/json" -d '{"id": 1, "descripcion": "Macarrones", "precio":0.87}' http://localhost:8080/productos
// curl http://localhost:8080/productos

// Busqueda de un producto por su id y luego por su descripcion
// curl http://localhost:8080/productos/1
// curl http://localhost:8080/productos/search/findByDescripcion?descripcion=Macarrones

// Modificar precio del producto y consulta para su verificacion
// curl -X PUT -H "Content-Type:application/json" -d '{"id": 1, "descripcion": "Macarrones", "precio":0.98}' http://localhost:8080/productos/1
// curl http://localhost:8080/productos/1

// Borrar un producto y consulta para su verificacion
// curl -X DELETE http://localhost:8080/productos/1
// curl http://localhost:8080/productos
```

## 3.c.RestController

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SaludoRest {

    // http://localhost:8080/hola
    @RequestMapping("/hola")
    public String hola() {
        return "Bienvenidos al curso";
    }

    // http://localhost:8080/adios?usuario="Anabel"
    @RequestMapping("/adios")
    public String adios(@RequestParam(value="usuario", defaultValue="Admin") String user) {
        return "Nos vamos a desayunar " + user;
    }
}
```

## 3.d. Lambdas



Definición:

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

## 3.d. Lambdas

### 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

## 3.d. Lambdas

### Contras

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

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

## 3.d. Lambdas

Como es una expresión lambda?

Forma básica:

arg  $\rightarrow$  body

Forma extendida:

( T1 arg1, T2 arg2 )  $\rightarrow$  { body }

# Misma definición, diferentes expresiones

```
new InterfaceA()
{
    public T2 method1(T1 v1)
    {
        return result;
    }
};
```



v1 -> result

# El ejemplo más esperado.. JButton!

## Con clases anónimas

```
button.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("click");
    }
}) ;
```

## Con lambdas

```
button.addActionListener(e -> System.out.println("click"));
```

# Inferencia de tipos, bloque de código, return

```
BinaryOperator<Integer> sumador= new BinaryOperator<Integer>()
{
    public Integer apply(Integer v1, Integer v2)
    {
        return v1 + v2;
    }
};
```



```
BinaryOperator<Integer> sumador=
```

```
(Integer v1, Integer v2) -> { return v1 + v2; }
```

ó

```
(v1, v2) -> { return v1 + v2; }
```

ó

```
(v1, v2) -> v1 + v2
```

# Interfaces funcionales

## Creadas a partir de lambdas

```
// Asignacion
Predicate<String> p = String::isEmpty;

// Predicate creado en contexto de la invocacion
stream.filter(e -> e.getSize() > 10)...

// En casteo
stream.map((ToIntFunction) e -> e.getSize())...

// Comparator creado con lambda
String[] list= new String[] { "One", "Two", "Three", "Four", "Five", "Six" };
Arrays.sort(list, (a, b)->a.substring(1).compareTo(b.substring(1)));
```

# Donde se pueden usar lambdas?

En contextos que tengan “target type”:

- Declaración de variables
- Asignaciones
- Instrucción “return”
- Inicializadores de Arrays
- Parámetros de métodos o constructores
- Cuerpo de un lambda
- Condicionales (?:)
- Casteos

Ejemplos para destacar:

```
FileFilter[] filters= new FileFilter[] { f -> f.exists(), f -> f.getName().startsWith("q") };  
  
Object o = (Runnable) () -> { System.out.println("hola"); };  
  
Supplier<Runnable> c = () -> () -> { System.out.println("hola"); };  
  
Callable<Integer> c = flag ? (() -> 23) : (() -> 42);
```

# Otra forma de lambda: referencia a método

## Tipos de referencias a métodos

- Método estático (ClassName::methName)
- Método de un instancia (instanceRef::methName)
- Método super de un objeto particular (super::methName)
- Método de instancia de un tipo (ClassName::methName)
- Constructor de clase (ClassName::new)
- Constructor de arreglo (TypeName[]::new)

## Ejemplos:

```
Collection<String> knownNames= Arrays.asList("Hello", "Java", "World", "8", "Streams");
Predicate<String> isKnown = knownNames::contains;

Function<String, String> upperfier = String::toUpperCase;

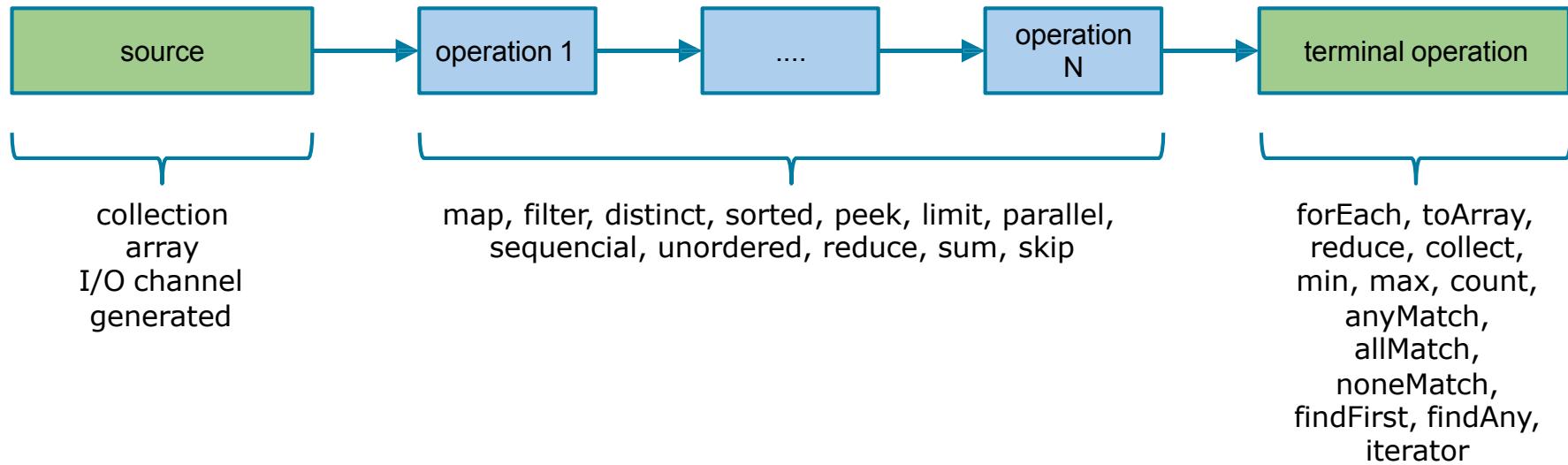
IntFunction<int[]> arrayMaker = int[]::new;
int[] array = arrayMaker.apply(10); // crea un arreglo int[10]
```

# Streams API

- 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.

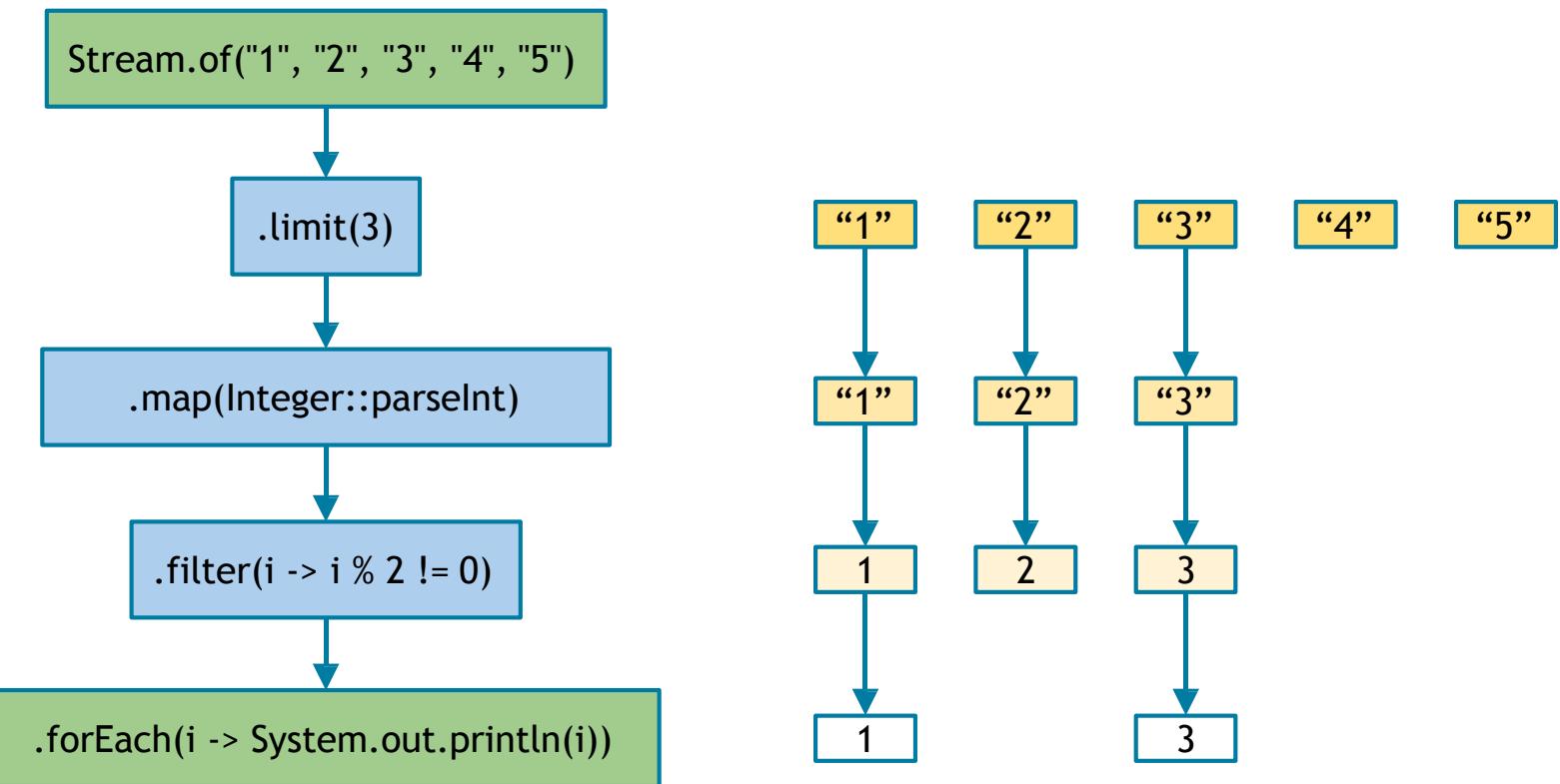
# Streams API

## Pipeline de operaciones



# Streams API

## Procesamiento de un stream



# Streams API

## Ejemplos

```
List<Integer> numbers = Arrays.asList(9, 10, 3, 4, 7, 3, -4);
List<Integer> distinct = numbers.stream().map(i -> i*i).distinct().collect(Collectors.toList());
```

```
IntStream.range(0, 100).parallel().filter(i -> i % 2 == 0).forEach(System.out::println);
```

```
List<String> G7 = Arrays.asList("USA", "Japan", "France", "Germany", "Italy", "U.K.", "Canada");
String G7Countries = G7.stream().map(x -> x.toUpperCase()).collect(Collectors.joining(", "));
```

```
List names = Arrays.asList("Java", "Scala", "C++", "Haskell", "Lisp");
Predicate<String> startsWithJ = n -> n.startsWith("J");
Predicate<String> fourLetterLong = n -> n.length() == 4;
names.stream()
.filter(startsWithJ.and(fourLetterLong))
.forEach((n) -> System.out.print("Empieza con J y tiene 4 caracteres: " + n));
```

# Streams API

## Ejemplos

```
Map<City, Set<String>> namesByCity= people.stream().collect(  
    Collectors.groupingBy (  
        Person::getCity, Collectors.mapping (Person::getLastName, toSet())  
    )  
) ;
```

*Resultado: Obtiene los apellidos de las personas en cada ciudad*

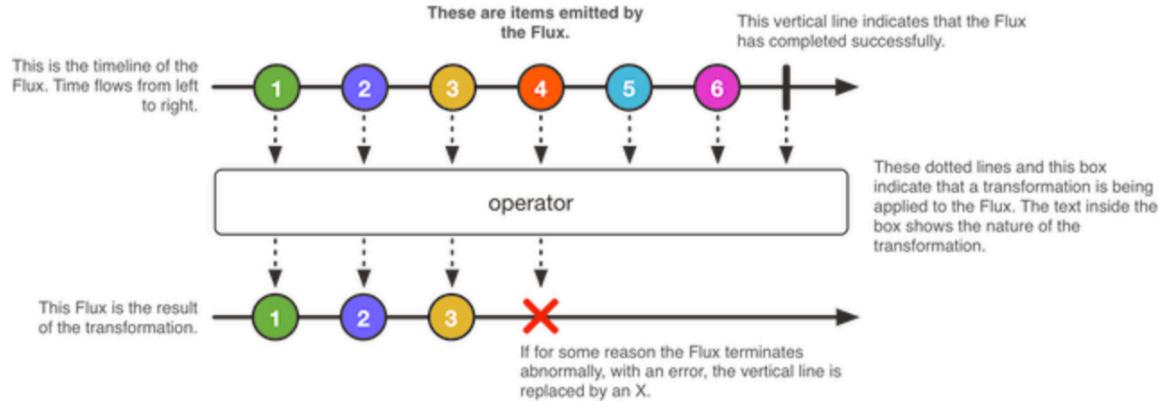
```
IntStream.iterate(0, i -> i + 2).limit(5).forEach(System.out::println);
```

*Resultado: imprime 0, 2, 4, 6, 8*

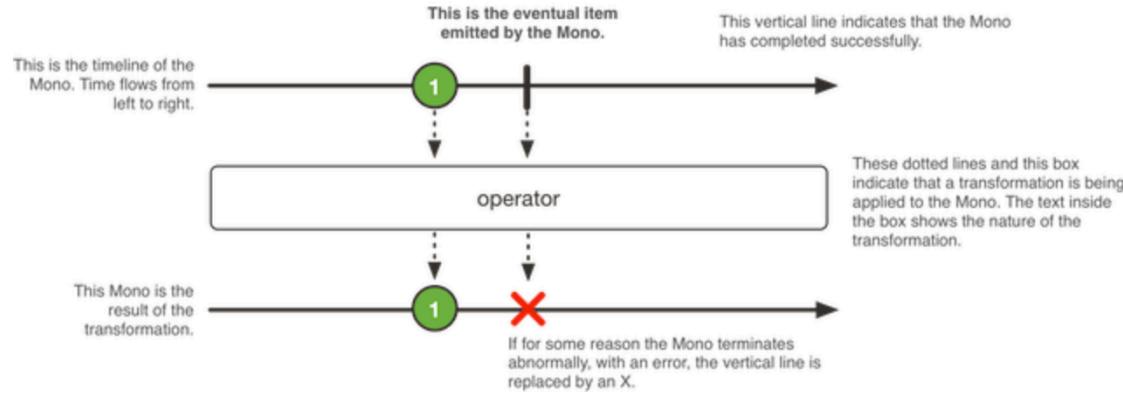
## 3.e. Mono y Flux



Flux:



Mono:



## 3.f. MongoDB

 MongoDB, a pesar de ser una base de datos relativamente joven (su desarrollo empezó en octubre de 2007) se ha convertido en todo un referente a la hora de usar bases de datos NoSQL y está listo para entornos de producción ágiles, de alto rendimiento y con gran carga de trabajo.

 En lugar de guardar los datos en tablas como se hace en las bases de datos relacionalas con estructuras fijas, las bases de datos NoSQL, como MongoDB, guarda estructuras de datos en documentos con formato JSON y con un esquema dinámico (MongoDB llama ese formato [BSON](#)).

 Ejemplo de documento almacenado en MongoDB:

```
{  
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),  
    "Last Name": "PELLERIN",  
    "First Name": "Franck",  
    "Age": 29,  
    "Address": {  
        "Street": "1 chemin des Loges",  
        "City": "VERSAILLES"  
    }  
}
```

## 3.f. MongoDB

-  Para descargar MongoDB debemos irnos a su pagina de descargas: <https://www.mongodb.org/downloads> donde encontrareis la versión adecuada a vuestra plataforma.
-  Una vez descargados los binarios de MongoDB para Windows, se extrae el contenido del fichero descargado (ubicado normalmente en el directorio de descargas) en C:\.
-  Renombra la carpeta a mongodb: C:\mongodb
-  MongoDB es autónomo y no tiene ninguna dependencia del sistema por lo que se puede usar cualquier carpeta que elijas. La ubicación predeterminada del directorio de datos para Windows es "C:\data\db". Crea esta carpeta.
-  Para iniciar MongoDB, ejecutar desde la Línea de comandos
  - C:\mongodb\bin\mongod.exe
-  Esto iniciará el proceso principal de MongoDB. El mensaje "waiting for connections" indica que el proceso mongod.exe se está ejecutando con éxito.

## 3.f. MongoDB



Dependencias necesarias:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

## 3.f. MongoDB

```
public class Producto {  
  
    @Id  
    private String id;  
  
    private String descripcion;  
    private double precio;  
  
    public String getDescripcion() {  
        return descripcion;  
    }  
  
    public void setDescripcion(String descripcion) {  
        this.descripcion = descripcion;  
    }  
  
    public double getPrecio() {  
        return precio;  
    }  
  
    public void setPrecio(double precio) {  
        this.precio = precio;  
    }  
}
```

## 3.g. Consultas a MongoDB

```
|  
@SpringBootApplication  
public class Application implements CommandLineRunner {  
  
    @Autowired  
    private ProductoRepository repository;  
  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
  
    @Override  
    public void run(String... args) throws Exception {  
  
        repository.deleteAll();  
  
        // alta de productos  
        repository.save(new Producto("Alargador", 18.34));  
        repository.save(new Producto("Bombilla Led", 5.23));  
  
        // listar todos los productos  
        System.out.println("Todos los productos encontrados");  
        System.out.println("-----");  
        for (Producto producto : repository.findAll()) {  
            System.out.println(producto);  
        }  
        System.out.println();  
  
        // Buscar un producto por su descripcion  
        System.out.println("Buscando Bombillas Led");  
        System.out.println("-----");  
        System.out.println(repository.findByDescripcion("Bombilla Led"));  
    }  
}
```

## 3.g. Consultas a MongoDB

```
// Levantar el servidor de mongo
// comando mongod

// Desde la consola con permisos de administrador

//Consultar todos los productos
// curl http://localhost:8080
// curl http://localhost:8080/productos

//Alta de producto y consulta para su verificacion
// curl -i -X POST -H "Content-Type:application/json" -d '{"descripcion": "Enchufe", "precio":5.32 }' http://localhost:8080/productos
// curl http://localhost:8080/productos

//Busqueda de un producto por su id y luego por su descripcion
// curl http://localhost:8080/productos/5d13409d5161a02ec107c7a9
// curl http://localhost:8080/productos/search
// curl http://localhost:8080/productos/search/findByDescripcion?descripcion=Enchufe

//Modificar precio del producto y consulta para su verificacion
// curl -X PUT -H "Content-Type:application/json" -d '{"descripcion": "Enchufe", "precio":7.12 }' http://localhost:8080/productos/5d13409d5161a02ec107c7a9
// curl http://localhost:8080/productos/5d13409d5161a02ec107c7a9

//Borrar un producto y consulta para su verificacion
// curl -X DELETE http://localhost:8080/productos/5d13409d5161a02ec107c7a9
// curl http://localhost:8080/productos
```

## 3.h. Repositorio MongoDB



Producto repository:

```
@RepositoryRestResource(collectionResourceRel = "productos", path = "productos")
public interface ProductoRepository extends MongoRepository<Producto, String> {
    List<Producto> findByLastName(@Param("descripcion") String descripcion);
}
```

# WebClient

Tema 4

## 4. WebClient

- a. Llamadas HTTP
- b. Request Body
- C. Filtros

## 4.a. Llamadas HTTP

## 4.b. Request Body

## 4.c. Filtros

# Spring Integration

Tema 5

# 5. Spring Integration

a. Introducción

b. Canales

C. Cliente

## 5.a. Introducción

## 5.b. Canales

## 5.c. Cliente

# Testing

Tema 6

# 6. Testing

- a. Test de Unidad
- b. Test de Integración
- C. Test de Aceptación
- d. Test de Carga
- e. Reporting

## 6.a. Test de Unidad

-  Pruebas Unitarias. Con ellas probamos las unidades del software, normalmente métodos.
-  Por ejemplo, escribimos estas pruebas para comprobar si un método de una clase funciona correctamente de forma aislada.
-  Por ejemplo, aunque estés probando un método que realice una serie de cosas y al final envíe un correo electrónico a través de un servidor de correo, en una prueba unitaria no tienes que probar que el correo se ha mandado correctamente.
-  Buenas pruebas unitarias no irían contra una base de datos, por ejemplo, sino que simularían la conexión.

## 6.b. Test de Integración

-  En este caso probamos cómo es la interacción entre dos o mas unidades del software.
-  Este tipo de pruebas verifican que los componentes de la aplicación funcionan correctamente actuando en conjunto.
-  Siguiendo con el caso anterior, las pruebas de integración son las que comprobarían que se ha mandado un email, la conexión real con la base de datos etc.
-  Este tipo de pruebas son dependientes del entorno en el que se ejecutan. Si fallan, puede ser porque el código esté bien, pero haya un cambio en el entorno.

## 6.c. Test de Aceptación



Las pruebas de aceptación se realizan para comprobar si el software cumple con las expectativas del cliente, con lo que el cliente realmente pidió.

## 6.d. Test de Carga



Las pruebas de carga son un tipo de prueba de rendimiento del sistema. Con ellas observamos la respuesta de la aplicación ante un determinado número de peticiones.



Aquí entraría por ejemplo ver cómo se comporta el sistema ante X usuarios que entran concurrentemente a la aplicación y realizan ciertas transacciones.

## 6.e. Reporting

# DOCUMENTACION



Herramientas utilizadas en el curso:

- ↳ JDK 8: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- ↳ Eclipse: <https://www.eclipse.org/downloads>
- ↳ Tomcat 8: <https://tomcat.apache.org/download-80.cgi>
- ↳ Maven: <https://maven.apache.org/download.cgi>



Sitio oficial de maven: <https://maven.apache.org>



Repositorio de maven: <https://mvnrepository.com>

Gracias por  
vuestra  
participación



¡Seguimos en contacto!

---

[www.iconotc.com](http://www.iconotc.com)

