





Objetivos y requisitos

- Objetivos del curso

- Entender los fundamentos de Spring: Inyección de dependencias y AOP. Configuración
- Diseñar aplicaciones con capas e interfaces bien definidas mediante la aplicación del patrón MVC
- Aprender la construcción de aplicaciones Web con el framework Spring MVC
- Aplicar las mejores prácticas para acceso a datos utilizando JPA (Hibernate) y JDBC
- Diseño y creación de controladores y control de errores
- Familiarizarse con los conceptos de programación reactiva con Reactor y Webflux
- Conocer los fundamentos de Spring Boot y cómo agiliza el desarrollo de aplicaciones web

- Requisitos

- Tener conocimientos del lenguaje Java

- Duración: 30 horas

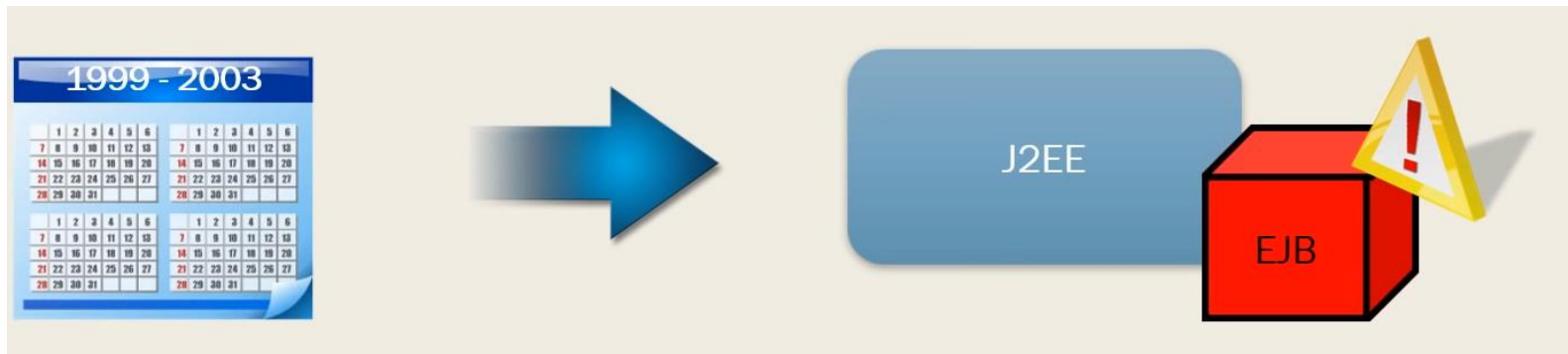


Temario

- Introducción a Spring y componentes
 - ¿Qué es Spring? Ventajas
 - Spring Boot – El problema y la solución
 - Preparación del entorno de desarrollo (JDK, STS, Maven)
 - Módulos de Spring
 - Beans y contenedores
 - Contextos de Spring
 - Patrón Singleton Vs Prototype
 - Inversión del control (IoC) e Inyección de dependencias (DI)
 - Expresiones Spring Expression Language (SpEL)
- Configuración
 - Configuración mediante ficheros XML (En desuso)
 - Configuración con anotaciones
- Anotaciones

¿Qué es Spring?

- Spring es un framework para el desarrollo de aplicaciones Java
- ¿Qué es un framework?
 - Es un “entorno de trabajo” compuesto por reglas y herramientas que facilitan enormemente el desarrollo y mantenimiento de aplicaciones
- Escrito por Rod Johnson en el año 2000 como alternativa al estándar EJB
 - Más simple y ligero que su homólogo en el desarrollo de aplicaciones JEE
- ¿Por qué se utiliza Spring?





Ventajas de Spring

- Inyección de dependencias (favorece el “loose coupling”)
- Desarrollo sencillo con POJOs (Plain Old Java Objects)
- Minimiza el código repetitivo (boilerplate code)
- Simplifica el acceso a datos
- Programación Orientada a Aspectos (AOP)
- Promueve una organización de la arquitectura en capas
- Fomenta la implementación de patrones de diseño y buenas prácticas



Spring Boot

- ¿Qué es Spring Boot?
 - El objetivo de Spring Boot es proporcionar un conjunto de herramientas para construir rápidamente aplicaciones de Spring que sean fáciles de configurar
- El problema: ¡Configurar Spring es complicado!
 - Las aplicaciones basadas en Spring tienen el inconveniente de que la mayor parte del trabajo consiste en configuraciones incluso para únicamente decir “Hola Mundo”
 - Esto no es algo malo: Spring es un elegante conjunto de infraestructuras que, para funcionar correctamente, requieren una configuración cuidadosamente coordinada. Pero, eso conlleva el costo de la complejidad en la configuración mediante complejos ficheros XML
- La solución: Spring Boot
 - Spring Boot facilita la creación de aplicaciones stand-alone basadas en Spring de grado de producción que « simplemente funcionan »
 - La mayoría de las aplicaciones Spring Boot necesitan una configuración mínima de Spring debido a la base de seguir la convención sobre la configuración. La poca configuración que se necesita está en forma de anotaciones en lugar de ficheros XML



Spring Boot

- Es intuitivo

- Spring Boot tiene criterios. Ésta es solo otra forma de decir que tiene valores predeterminados razonables que implementan las configuraciones más utilizadas por la mayor parte de desarrolladores
- Por ejemplo, Tomcat es un contenedor web muy popular. De forma predeterminada, una aplicación web de Spring Boot utiliza un contenedor Tomcat incorporado y listo para ser ejecutado.

- Es personalizable

- Una infraestructura intuitiva no es de mucha utilidad si no es posible cambiar sus criterios. Es posible personalizar fácilmente una aplicación Spring Boot para que coincida con las necesidades del proyecto, tanto en configuración inicial como en el ciclo de desarrollo

- Iniciadores (Starters)

- Los iniciadores son parte de la magia de Spring Boot, se utilizan para limitar la cantidad de configuración manual de las dependencias. Un iniciador es esencialmente un conjunto de dependencias (como un Maven POM) que son específicas para el tipo de aplicación que representa
- Spring Boot utiliza la manera en que se definen los Beans para configurarse automáticamente. Por ejemplo si anotamos beans de JPA con `@Entity`, Spring Boot configurará automáticamente JPA sin necesidad de un archivo `persistence.xml`
- Ejemplos: `Spring boot-starter-web`, `spring-boot-starter-jdbc`












Preparación del Entorno de Desarrollo

- Instalación de JRE y JDK de Java (v. 11+)
 - JDK: <https://www.oracle.com/java/technologies/downloads/#javasejdk>
 - JRE: https://www.java.com/es/download/ie_manual.jsp
- Instalación de Maven (v. 3.1+)
 - <https://maven.apache.org/download.cgi>
 - <https://maven.apache.org/install.html>
- Instalación de Spring Tool Suite (v. 4.1+)
 - <https://spring.io/tools>

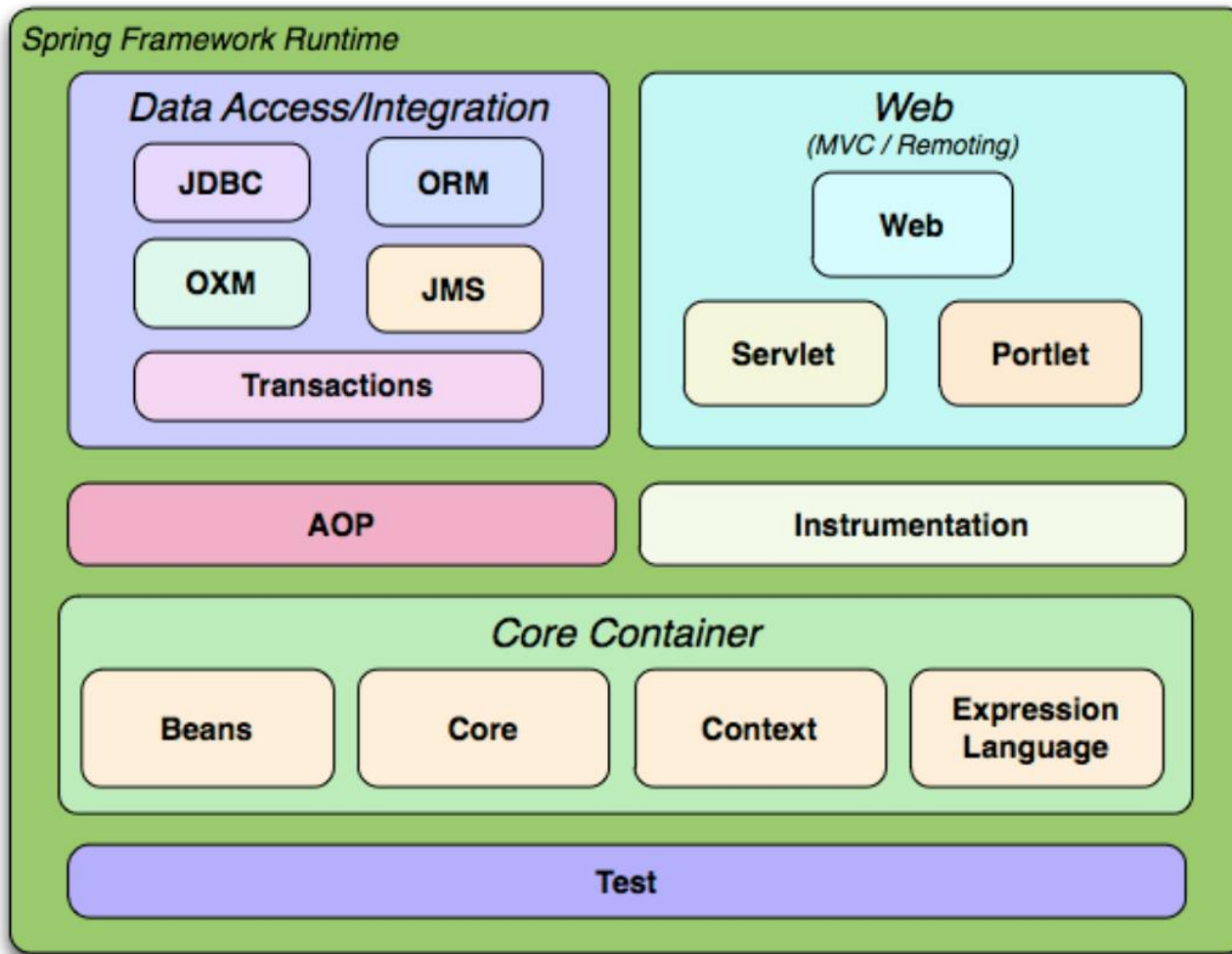


Módulos Spring

 <p>SPRING BOOT</p> <p>Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.</p>	 <p>SPRING FRAMEWORK</p> <p>Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.</p>	 <p>SPRING XD</p> <p>Simplifies the development of big data applications by addressing ingestion, analytics, batch jobs and data export.</p>	 <p>SPRING DATA</p> <p>Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.</p>	 <p>SPRING INTEGRATION</p> <p>Supports the well-known <i>Enterprise Integration Patterns</i> via lightweight messaging and declarative adapters.</p>	 <p>SPRING BATCH</p> <p>Simplifies and optimizes the work of processing high-volume batch operations.</p>
 <p>SPRING SECURITY</p> <p>Protects your application with comprehensive and extensible authentication and authorization support.</p>	 <p>SPRING HATEOAS</p> <p>Simplifies creating REST representations that follow the HATEOAS principle.</p>	 <p>SPRING SOCIAL</p> <p>Easily connects your applications with third-party APIs such as Facebook, Twitter, LinkedIn, and more.</p>	 <p>SPRING AMQP</p> <p>Applies core Spring concepts to the development of AMQP-based messaging solutions.</p>	 <p>SPRING MOBILE</p> <p>Simplifies the development of mobile web apps through device detection and progressive rendering options.</p>	 <p>SPRING FOR ANDROID</p> <p>Provides key Spring components for use in developing Android applications.</p>
 <p>SPRING WEB FLOW</p> <p>Supports building web applications with controlled navigation such as checking in for a flight or applying for a loan.</p>	 <p>SPRING WEB SERVICES</p> <p>Facilitates the development of contract-first SOAP web services.</p>	 <p>SPRING LDAP</p> <p>Simplifies the development of applications using LDAP using Spring's familiar template-based approach.</p>	 <p>GRAILS</p> <p>Builds on Spring to provide a full-stack environment for creating web applications using the Groovy language.</p>	 <p>GROOVY</p> <p>Brings high-productivity language features to the JVM including support for static and dynamic programming, scripting, and domain-specific languages.</p>	 <p>SPRING CLOUD</p> <p>Simplifies connecting to services and gaining operating environment awareness in cloud platforms like Cloud Foundry and Heroku.</p>



Módulos Spring





¿Qué son los Beans?

- Un Java Bean o simplemente Bean, es una clase simple en Java que cumple ciertas normas con los nombres de sus propiedades y métodos. Es un componente (similar a una caja) que nos permite encapsular el contenido, con la finalidad de otorgarle una mejor estructura y además permite la reutilización de código
- Se aconseja que un Bean cumpla las siguientes condiciones:
 - Constructor sin argumentos: Aunque existe por defecto se recomienda declararlo
 - Atributos privados
 - Getters y Setters de todos los atributos
 - Puede implementar serializable
 - Requiere una anotación o crear un fichero .xml donde se detalle que es un Bean

¿Qué es Spring Container?

- Es el núcleo del framework Spring.
- El contenedor de Spring es el encargado de crear objetos (Beans), unirlos, configurarlos y administrar su ciclo de vida





Contextos de Spring

- El Spring Context no es más que un contenedor IoC de Spring que proporciona un ámbito a los Beans que aloja
- Ejemplos de contextos en Spring:
 - ApplicationContext
 - WebApplicationContext
 - SecurityContext
 - ServletContext



Patrón Singleton

- ¿Qué es?
 - Patrón de diseño que tiene como objetivo asegurar que solo hay una instancia u objeto por clase y un punto de acceso global a ella. **Spring utiliza Singleton por defecto**
- ¿Por qué es necesario ésto?
 - En determinados escenarios que surgen a la hora de programar, debemos asegurarnos de que las clases controlan y gestionan el acceso a un único recurso (p. ej. Un fichero abierto). Es frecuente necesitar un punto de acceso único a algún recurso del sistema compartido y un único objeto que centralice la administración del recurso
 - En otros escenarios la necesidad es que diferentes objetos de la aplicación puedan acceder a un tipo de dato común en concreto
- Ventajas de utilizar este patrón
 - Control estricto de cómo se acceden a las instancias
 - Espacio de nombres reducido (mayor claridad de código)
 - Ahorro de recursos del sistema



Patrón Prototype

- ¿Qué es?
 - Patrón de diseño que tiene como objetivo la creación de varios objetos a partir de un modelo o “prototipo”. Esto lo hace mediante la clonación de objetos o instancias creadas previamente. Siempre es más óptimo clonar un objeto que crear uno nuevo. El objeto tendrá sus propios valores desde setters
- ¿Por qué es necesario ésto?
 - En determinados escenarios que surgen a la hora de programar se necesitan varios objetos con atributos repetidos (comunes)
- Ventajas de utilizar este patrón
 - Una aplicación puede crear y eliminar objetos en tiempo de ejecución
 - Permite crear nuevos objetos variando los ya existentes con el consiguiente ahorro de tiempo y recursos
 - Clonar siempre es más rápido que crear



Definición de Beans

- Mediante XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

  <bean id="secretario" class="org.cursospringboot.model.Secretario">
  </bean>
  <bean id="director" class="org.cursospringboot.model.Director">
  </bean>
  <bean id="jefe" class="org.cursospringboot.model.Jefe">
  </bean>
</beans>
```

- Mediante anotaciones

- @Bean
- @Component o @Service
- @Entity
- @Controller

```
@Bean("jefe")
public Empleado getJefe() {
    return new Jefe();
}

@Bean("director")
public Empleado getDirector() {
    return new Director();
}

@Bean("secretario")
public Empleado getSecretario() {
    return new Secretario();
}
```




Dependencias entre Beans

- Generalmente los Beans necesitan de otros Beans para ejecutar sus tareas, para ello Spring dispone de alternativas a la hora de definir dichas dependencias según las necesidades o preferencias del programador
- Varios formas de realizar el “cableado” o “wiring”
 - Propiedades: Se utiliza una propiedad de una clase para definir la dependencia
 - “Setters”: Spring permite la configuración de una dependencia mediante la función `setProperty`
 - Constructor: La dependencia se configura en el constructor de la clase inyectando así los argumentos de dicho método
- Tipos de configuración
 - Mediante fichero XML
 - Mediante la anotación `@Autowired`

Configuración de cableado mediante XML

- Propiedades

```
<bean id="jefe" class="org.cursospringboot.model.Jefe">  
    <property name="email" value="jefe@email.com"></property>  
</bean>
```

- Setter

```
<bean id="generadorInformes" class="org.cursospringboot.model.GeneradorInformes"></bean>  
  
<bean id="director" class="org.cursospringboot.model.Director">  
    <property name="informe" ref="generadorInformes"></property>  
</bean>
```

- Constructor

```
<bean id="generadorInformes" class="org.cursospringboot.model.GeneradorInformes"></bean>  
  
<bean id="secretario" class="org.cursospringboot.model.Secretario">  
    <constructor-arg ref="generadorInformes"></constructor-arg>  
</bean>
```

Configuración de cableado mediante @Autowired

- Spring escaneará todas las clases del paquete base en busca de anotaciones @Autowired y tratará de encontrar un Bean adecuado en el contenedor de Beans
- Modos de Autowiring
 1. Por nombre: Utilizando el nombre de la propiedad, el contenedor de Spring buscará un Bean cuyo nombre coincida con ésta
 2. Por tipo: Utilizando el tipo de la propiedad, el contenedor buscará un Bean cuyo tipo coincida con el mismo
- El contenedor Spring tratará de encontrar un bean por nombre y después por tipo. Si no consigue encontrarlo o existen varias coincidencias se lanzará una excepción al inicial la aplicación
- Resolución de conflictos en @Autowired con @Qualifier
- Podremos definir el ámbito de los beans mediante la anotación @Scope

– @Scope("prototype")	@Bean("generadorDeInformes")
	@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
– @Scope("session")	public GeneradorInformes getGeneradorInformes() {
	return new Informe();
	}



Inversión de Control (IoC)

- Principio de Hollywood (Martin Fowler)



- Ventajas
 - Proporciona modularidad
 - Permite ampliar la funcionalidad de nuestras aplicaciones sin modificar las clases
 - Evita la dependencia entre las clases
 - Flexibiliza nuestras aplicaciones haciéndolas más configurables