Introducción a Spring Framework

El framework de Java más utilizado para desarrollar aplicaciones empresariales

 **por Ivan Ruiz Rube**

Agenda de este tema

Fundamentos de Spring

Introducción al framework, historia y conceptos básicos de Spring.

Arquitectura Spring

Contenedores, beans, componentes e inyección de dependencias.

Configuración y AOP

Métodos de configuración y programación orientada a aspectos.

Spring Boot

Starters, Spring Initializr y desarrollo rápido de aplicaciones.

Implementación Práctica

Ejemplos prácticos y primeros pasos con Spring.

Spring Framework

Spring Framework es una plataforma de código abierto para Java creada en 2002 por Rod Johnson y mantenido inicialmente por la empresa SpringSource, luego Pivotal y desde 2019 VMware. Se ha consolidado como el ecosistema líder para desarrollar aplicaciones empresariales, destacando por su arquitectura modular, bajo acoplamiento y filosofía de "convención sobre configuración".

Hitos Spring

1

Marzo 2004 – Spring Framework 1.0

Primera versión final: principios IoC, AOP, módulos de JDBC/JTA, consolidando el modelo de programación en Java más ligero que con EJB.

2

Mayo 2006 – Java EE 5

Llega con anotaciones (`@Entity`, `@EJB`, etc.) y JPA, simplificando drásticamente la complejidad de desarrollo en la plataforma Java EE.

3

Diciembre 2009 – Spring 3.0

Introduce SpEL (Spring Expression Language), JavaConfig nativo (configuración solo con clases Java) y soporte REST integrado, alineándose con tendencias web.

4

Diciembre 2009 – Java EE 6

Añade CDI (Contexts & Dependency Injection), JAX-RS (REST) y Bean Validation; orienta Java EE hacia arquitecturas basadas en componentes y servicios web.

5

Abril 2014 – Spring Boot 1.0

Revoluciona Spring con “starters” y autoconfiguración: levanta apps completas en segundos eliminando XML y simplificando dependencias.

6

Septiembre 2017 – Spring 5.0

Soporta Java 9, estrena el stack reactivo (Spring WebFlux) y se moderniza con programación asíncrona y no bloqueante.

7

Septiembre 2019 – Jakarta EE 8

Primer lanzamiento bajo Eclipse Foundation (idéntico a Java EE 8, pero con paquete `jakarta.*`), marcando el cambio de gobernanza de la plataforma.

8

Diciembre 2020 – Jakarta EE 9

Realiza el cambio total de namespace de `javax.*` a `jakarta.*`, sentando las bases para la evolución futura de Jakarta EE.

9

Noviembre 2022 – Spring 6.0 / Spring Boot 3.0

Requieren Java 17, adoptan `jakarta.*`, optimizan para AOT/GraalVM Native y alinean todo el ecosistema Spring con el nuevo namespace.

10

Marzo 2025 – Jakarta EE 11 Web Profile

Alineada con Java 21, modularizada y con CDI 4.0 mejorado; define el perfil moderno para microservicios y aplicaciones empresariales ligeras.

Jakarta EE vs Spring

Jakarta EE

Java EE (ahora *Jakarta EE*) es la plataforma estándar de Java empresarial (API oficiales para web, EJB, JPA, JSF, etc.)

Enfoque basado en estándares oficiales con especificaciones formales (Java Community Process)

Implementaciones por diferentes proveedores que deben cumplir con las especificaciones

Spring

Ecosistema alternativo *open-source* con un enfoque más ligero y modular

Mayor flexibilidad y evolución más rápida al no estar limitado por procesos de estandarización y estár dirigido por una única empresa

Enfoque pragmático orientado a la productividad del desarrollador

Contenedor Tradicional vs Contenedor Ligero

Java EE: Servidor de Aplicaciones

En Java EE, las aplicaciones se despliegan en servidores de aplicaciones Java (WildFly, GlassFish, WebLogic...)

El servidor proporciona todas las implementaciones de las API y de los servicios de la plataforma Jakarta EE.

Múltiples aplicaciones comparten el mismo servidor

Configuración compleja

Spring Boot: Servidor Embebido

Spring Boot empaqueta la aplicación con un servidor web embebido

Elimina la necesidad de un contenedor Java EE externo

Agiliza pruebas y despliegues en la nube

Las apps deben incluir todos los componentes (jar) requeridos

Se facilita el aislamiento y la independencia entre apps

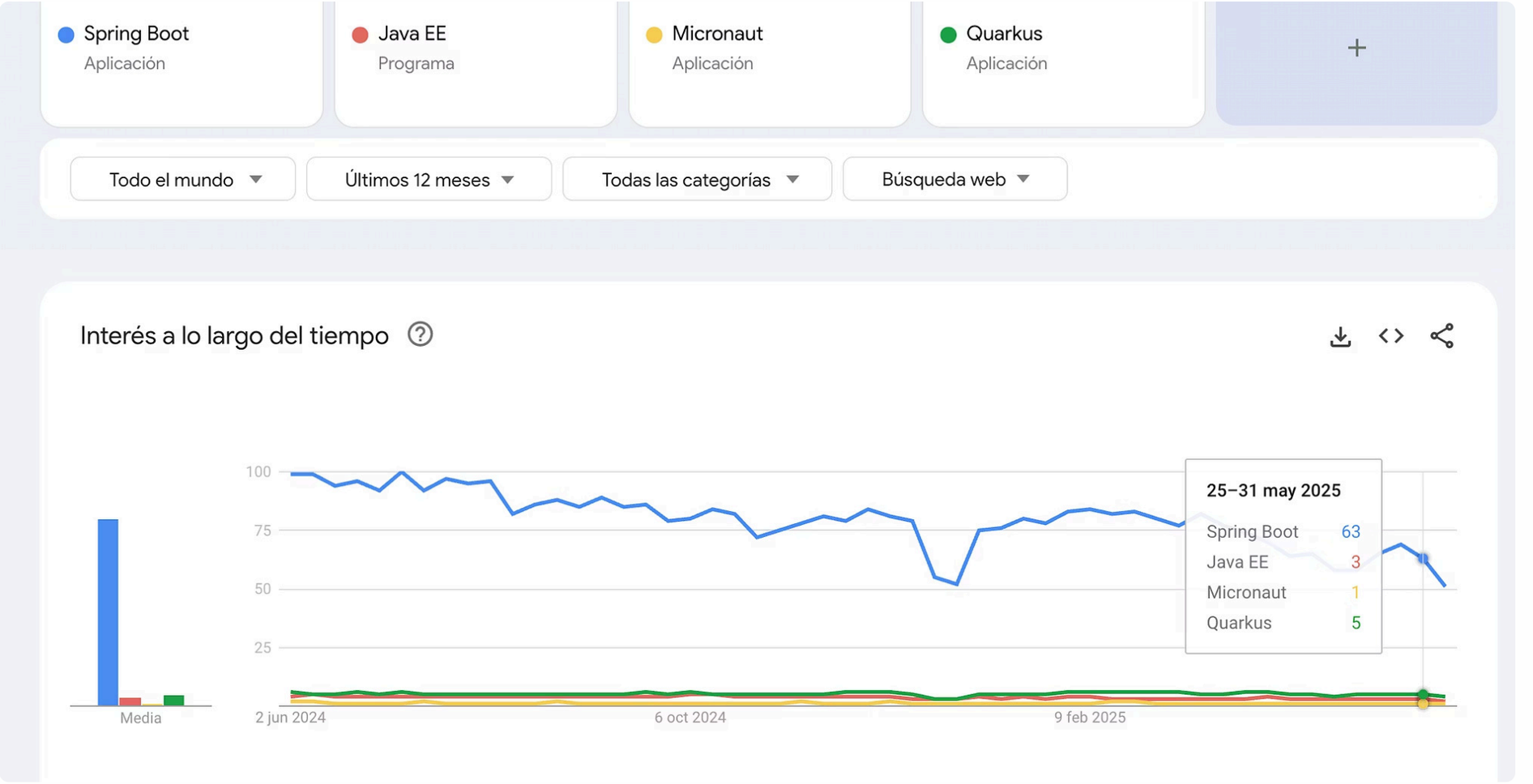
Facilita la adopción de prácticas DevOps

Equivalencias entre Jakarta EE y Spring Framework

Estándar Jakarta EE	Equivalente en Spring
Jakarta Servlet	Spring Web MVC (`DispatcherServlet`, controladores `@Controller`/`@RestController`)
Jakarta Expression Language (EL)	Spring Expression Language (SpEL) (`@Value`, `@Conditional`, `ExpressionParser`)
Jakarta Contexts & Dependency Injection (CDI)	Contenedor de Inversión de Control (IoC) de Spring (`@Component`, `@Service`, `@Autowired`)
Jakarta Persistence (JPA)	Spring Data JPA (repositorios `JpaRepository`, `EntityManager` gestionado por Spring)
Jakarta Bean Validation	Spring Validation (Hibernate Validator + `@Valid`, `Validator` de Spring)
Jakarta Transaction (JTA)	Spring Transaction Management (`@Transactional`, `PlatformTransactionManager`)
Jakarta Batch	Spring Batch (`Job`, `Step`, `JobLauncher`, `ItemReader/ItemWriter`)
Jakarta Mail (JavaMail)	Spring Mail (`JavaMailSender`, configuración `spring.mail.*`)
Jakarta WebSocket	Spring WebSocket (`WebSocketConfigurer`, `@EnableWebSocket`, STOMP sobre WebSocket)
Jakarta JSON Processing (JSON-P)	Soporte de Jackson (incluido en Spring Boot) vía `MappingJackson2HttpMessageConverter`
Jakarta JSON Binding (JSON-B)	Jackson o Gson (configurados automáticamente en Spring Boot para (de)serialización)
Jakarta JAX-RS (REST)	Spring Web MVC REST (`@RestController`, `@RequestMapping`) o Spring WebFlux (`@RestController` reactivo)
Jakarta JMS	Spring JMS (`JmsTemplate`, `DefaultJmsListenerContainerFactory`)
Jakarta Mail	Spring Mail (`JavaMailSender`, `MailSender`)
Jakarta Validation	Spring Validation (`Validator`, `@Validated`, `@Valid`)
Jakarta Security	Spring Security (`WebSecurityConfigurerAdapter`/`SecurityFilterChain`, `@EnableWebSecurity`)
Jakarta Enterprise Beans (EJB)	Equivalentes parciales: Spring Managed Beans (`@Service`, `@Transactional`), Spring Remoting, Spring's tareas programadas (`@Scheduled`)
Jakarta Connectors (JCA)	Spring Resource Adapters (uso de `JcaContainerConnectionFactory`, integración JCA)
Jakarta Naming & Directory (JNDI)	Spring JNDI (`JndiObjectFactoryBean`, `JndiDataSourceLookup`)
Jakarta Management (JMX)	Spring JMX (`MBeanExporter`, `@ManagedResource`, `@ManagedAttribute`)

Esta equivalencia facilita la migración entre ambas tecnologías y muestra cómo Spring proporciona alternativas para prácticamente todas las funcionalidades de Jakarta EE.

Tendencias en frameworks Java



Spring Boot ha ganado enorme popularidad por su productividad y ecosistema unificado. Jakarta EE sigue presente, especialmente en entornos legacy y aplicaciones desplegadas en servidores de aplicaciones tradicionales. Quarkus está empezando a ganar peso en arquitecturas cloud-native y microservicios ultraligeros.

Comunidad y Soporte de Spring

Amplia adopción

Spring es ampliamente utilizado en la industria, con gran comunidad

Documentación extensa

Guías de referencia, tutoriales y ejemplos detallados. Actualizada regularmente

Soporte comercial

VMware ofrece soporte comercial para entornos empresariales: corrección de errores, formación, consultoría, etc.

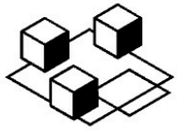
Lanzamientos frecuentes

Actualizaciones regulares alineadas con las versiones de Java

¿Qué podemos hacer con Spring?

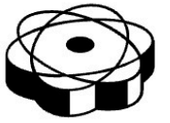
Generative AI

Integrate AI into your Spring applications without reinventing the wheel.



Microservices

Quickly deliver production-grade features with independently evolvable microservices.

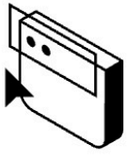


Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.

Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.

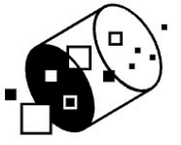


Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.

Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.



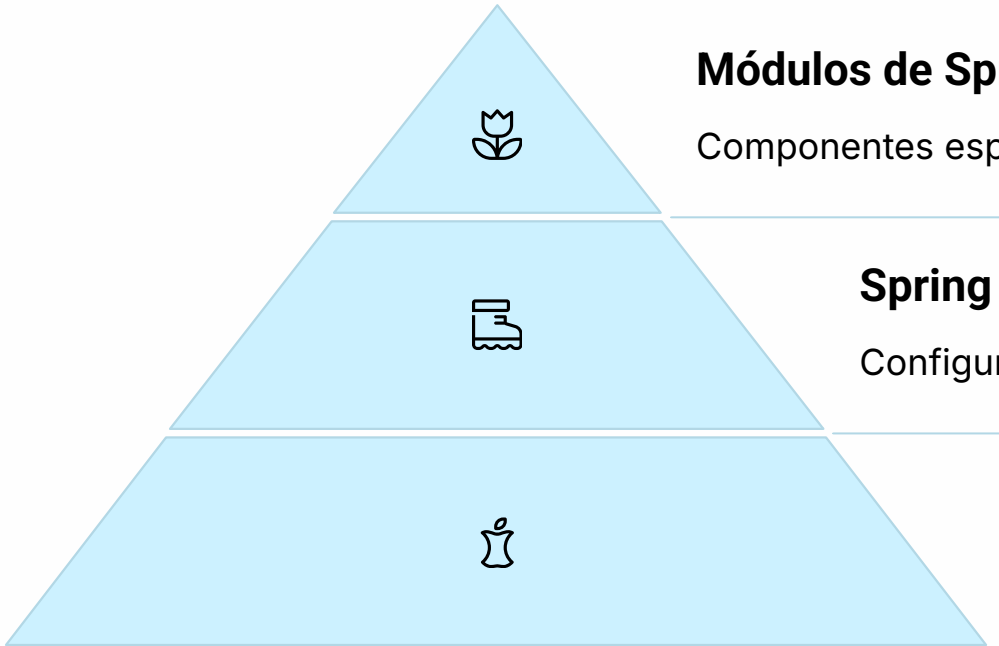
Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.

Batch

Automated tasks. Offline processing of data at a time to suit you.

Ecosistema Spring



Módulos de Spring

Componentes especializados para tareas específicas

Spring Boot

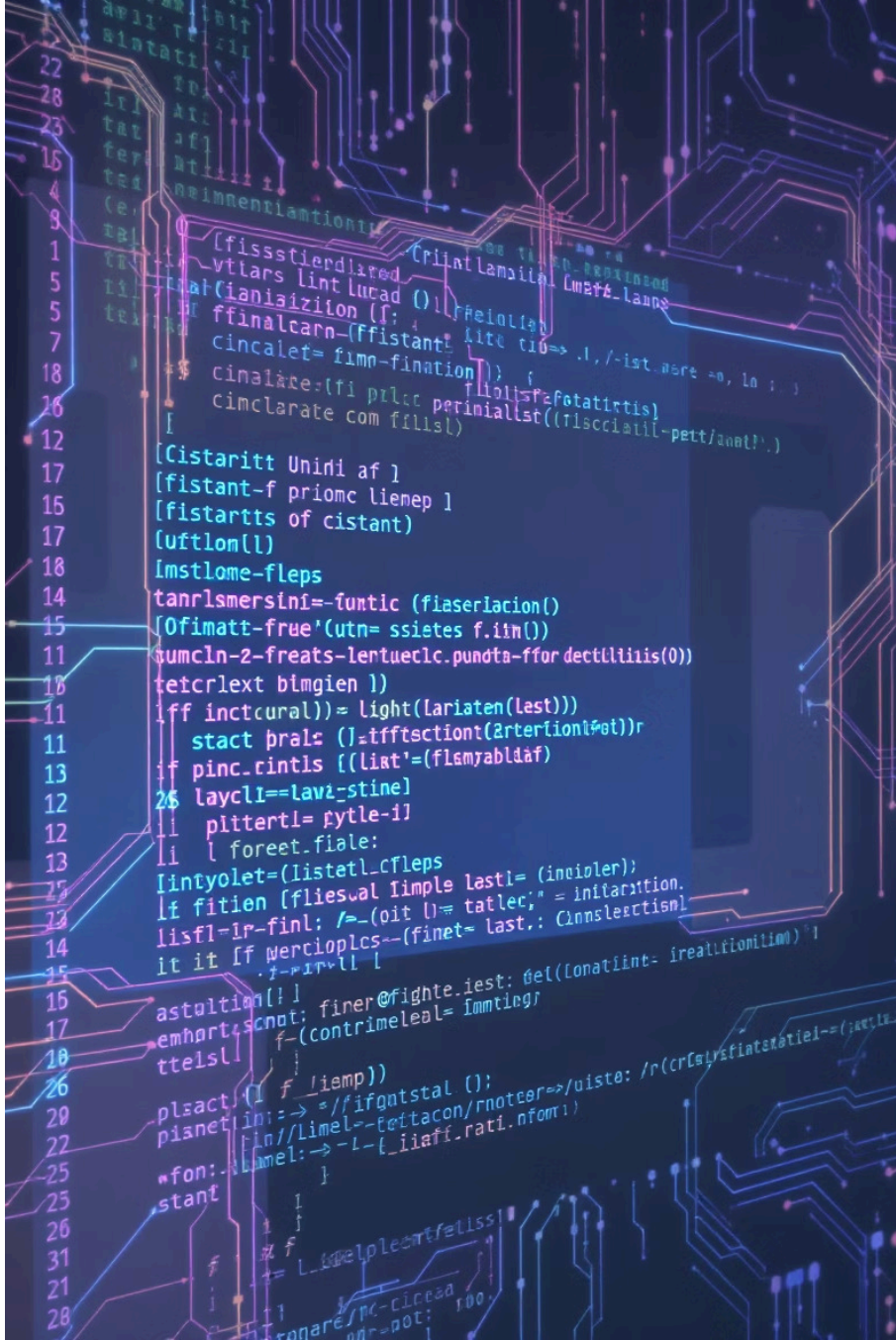
Configuración automática + apps autoarrancables

Spring Framework Core

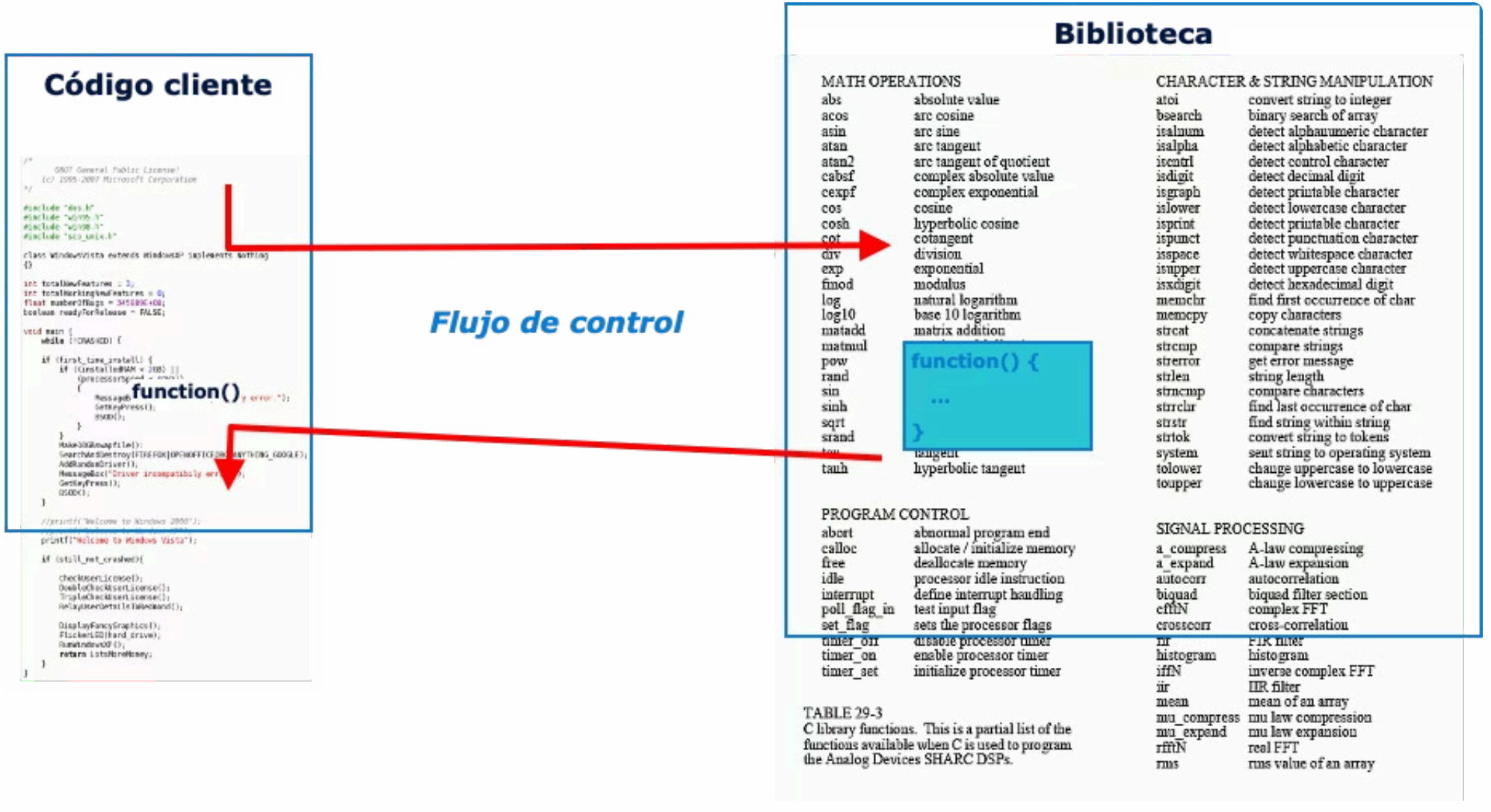
Contexto de aplicación, IoC, DI, AOP

Pilares de Spring

1. Inversión de Control
2. Inyección de Dependencias
3. Contenedor de Beans
4. Orientación a Aspectos



¿Spring es una biblioteca?



MATH OPERATIONS

abs	absolute value
acos	arc cosine
asin	arc sine
atan	arc tangent
atan2	arc tangent of quotient
cabsf	complex absolute value
cexpf	complex exponential
cos	cosine
cosh	hyperbolic cosine
cot	cotangent
div	division
exp	exponential
fmod	modulus
log	natural logarithm
log10	base 10 logarithm
matadd	matrix addition
matmul	
pow	
rand	
sin	
sinh	
sqrt	
srand	
tan	tangent
tanh	hyperbolic tangent

CHARACTER & STRING MANIPULATION

atoi	convert string to integer
bsearch	binary search of array
isalnum	detect alphanumeric character
isalpha	detect alphabetic character
iscntrl	detect control character
isdigit	detect decimal digit
isgraph	detect printable character
islower	detect lowercase character
isprint	detect printable character
ispunct	detect punctuation character
isspace	detect whitespace character
isupper	detect uppercase character
isxdigit	detect hexadecimal digit
memchr	find first occurrence of char
memcpy	copy characters
strcat	concatenate strings
strcmp	compare strings
strerror	get error message
strlen	string length
strncmp	compare characters
strrchr	find last occurrence of char
strstr	find string within string
strtok	convert string to tokens
system	sent string to operating system
tolower	change uppercase to lowercase
toupper	change lowercase to uppercase

PROGRAM CONTROL

abort	abnormal program end
calloc	allocate / initialize memory
free	deallocate memory
idle	processor idle instruction
interrupt	define interrupt handling
poll_flag_in	test input flag
set_flag	sets the processor flags
timer_off	disable processor timer
timer_on	enable processor timer
timer_set	initialize processor timer

SIGNAL PROCESSING

a_compress	A-law compressing
a_expand	A-law expansion
autocorr	autocorrelation
biquad	biquad filter section
cfftN	complex FFT
crosscorr	cross-correlation
fir	FIR filter
histogram	histogram
ifftN	inverse complex FFT
iir	IIR filter
mean	mean of an array
mu_compress	mu law compression
mu_expand	mu law expansion
rfftN	real FFT
rms	rms value of an array

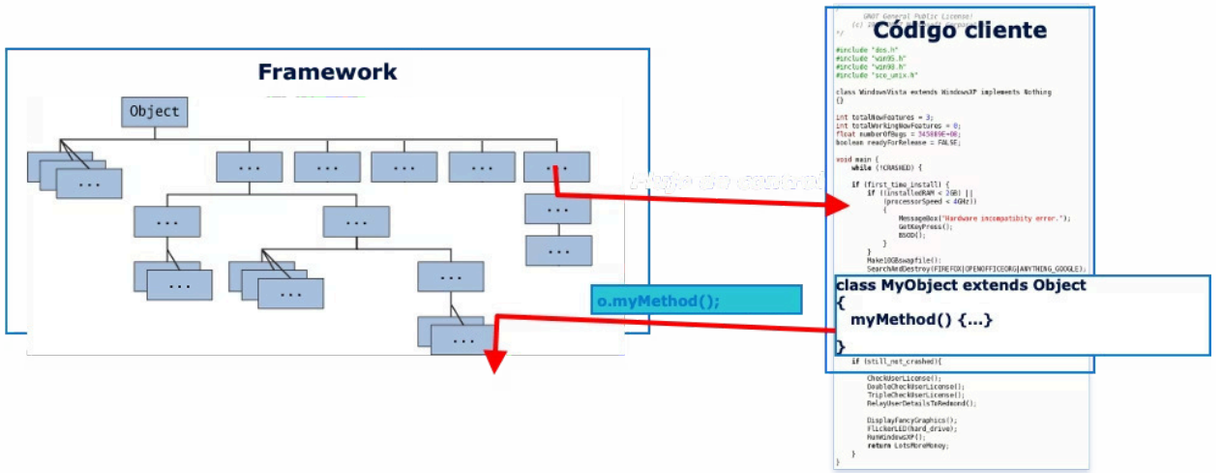
TABLE 29-3
C library functions. This is a partial list of the functions available when C is used to program the Analog Devices SHARC DSPs.

Spring es un Framework

Inversión de Control (IoC)

Principio de hollywood

El control del flujo del programa se invierte. En lugar de que el código de la aplicación llame al framework, el **framework** llama al código de la aplicación.



Contenedor de Beans

El contenedor de beans es el corazón de Spring, define el contexto de la aplicación y centraliza la gestión de objetos.

Gestión del Ciclo de Vida

Creación (@PostConstruct), configuración y destrucción (@PreDestroy) de objetos

Alcance

Singleton (una instancia por contexto, predeterminado), prototype (nueva instancia cada vez), request (una instancia por petición HTTP), session (una instancia por sesión de usuario)

Registro y Configuración

XML, en Java (@Configuration) o vía anotaciones (@Component)

Desarrollo basado en componentes

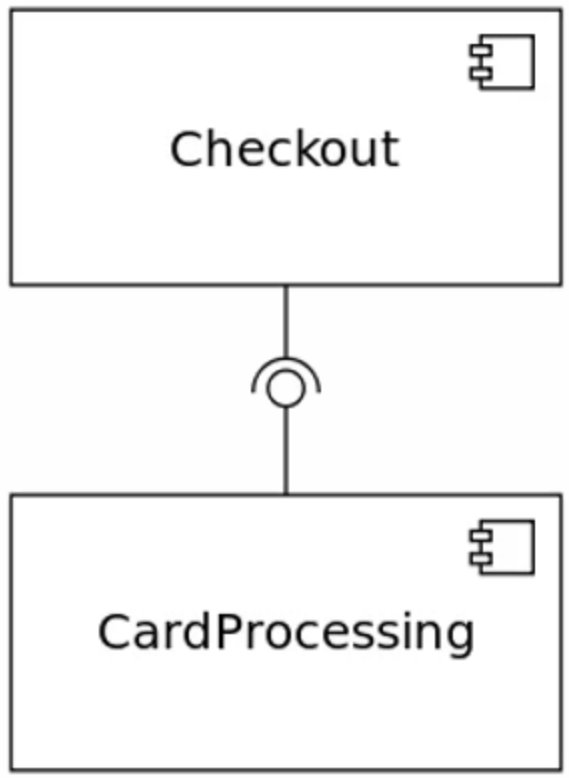
El desarrollo de aplicaciones modernas requiere gestionar eficientemente la interconexión entre múltiples componentes.

Desafíos

La gestión del ciclo de vida de objetos plantea desafíos: ¿quién debe ser responsable de instanciar, configurar y conectar los componentes necesarios?

Cableado

La interconexión entre componentes debe establecerse minimizando el acoplamiento, favoreciendo así la flexibilidad y facilitando el mantenimiento a largo plazo.



Interfaces

El uso de interfaces como contratos bien definidos permite sustituir implementaciones sin afectar a los consumidores, simplificando también las pruebas unitarias.

Problema

A pesar del uso de interfaces, los objetos requieren instanciar los objetos dependientes.

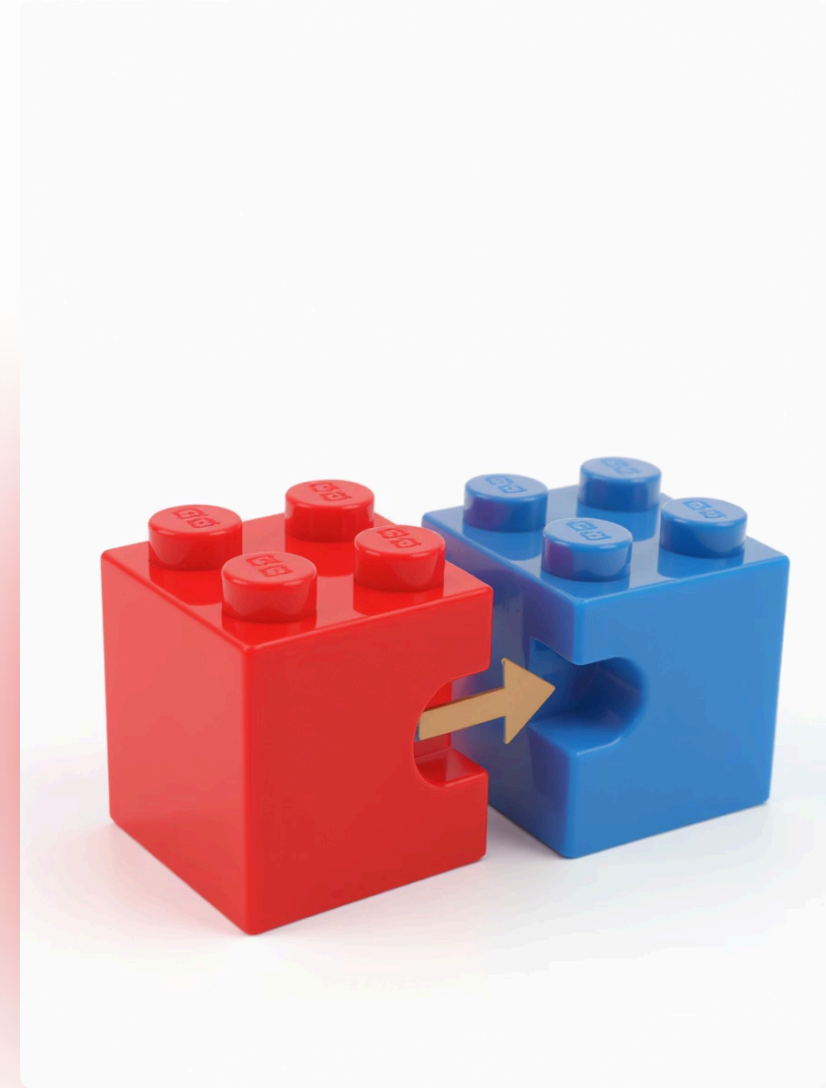
Spring actúa como ensamblador (assembler) externo que gestiona la creación y conexión (inyección) de objetos, liberando a las clases de la responsabilidad de crear sus propias dependencias.

Inyección de Dependencias (DI)

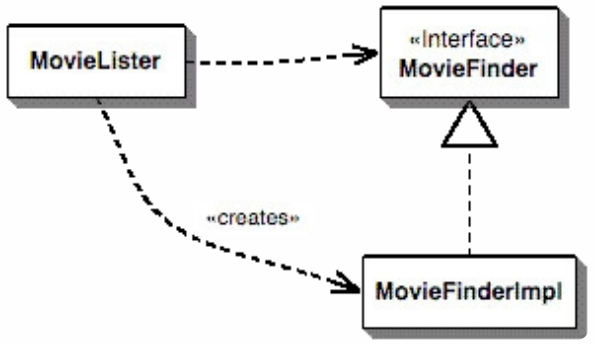
Técnica por la cual los objetos reciben sus dependencias en lugar de crearlas, eliminando el uso directo del operador "new" en el código.

Esta técnica proporciona tres beneficios principales:

- Desacoplamiento entre componentes
- Mayor facilidad para realizar pruebas unitarias
- Configuración dinámica de la aplicación



Sin inyección de dependencias

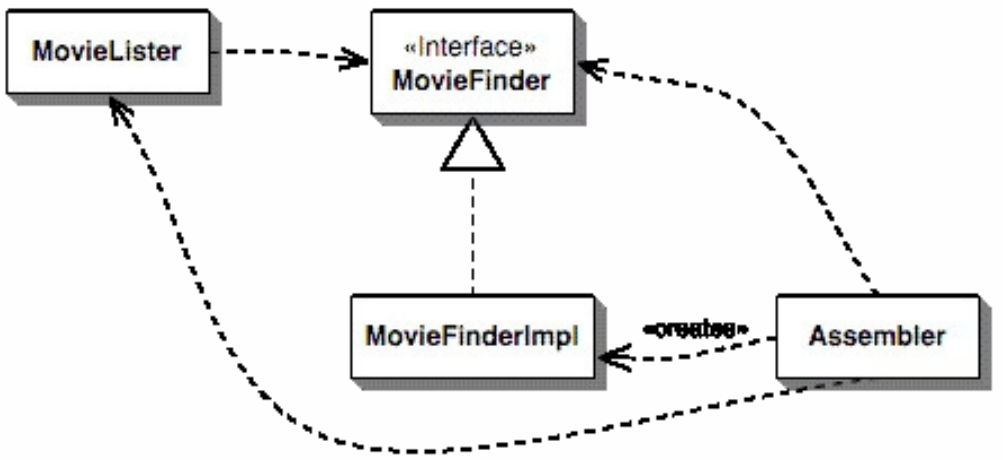



```
// Ejemplo de Martin Fowler

public class MovieLister {

        private MovieFinder finder;

        public MovieLister() {
                this.finder = new MovieFinderImpl();
        }

        public List<Movie> getMoviesDirectedBy(String director) {
                // No hagan esto en casa ;-)

                List<Movie> result = new ArrayList<Movie>();
                List<Movie> allMovies = finder.findAll();

                for (Movie movie : allMovies) {
                        if (movie.getDirector().equals(director)) {
                                result.add(movie);
                        }
                }
                return result;
        }


        public List<Movie> getAllMovies() {
                return finder.findAll();
        }
}
```

Con inyección de dependencias

Métodos de inyección:

- **Por constructor:** Método recomendado para dependencias obligatorias. Garantiza objetos completamente inicializados desde su creación y favorece la inmutabilidad de los componentes.
- **Por setters:** Ideal para dependencias opcionales o cuando se necesita reemplazar o modificar dependencias en tiempo de ejecución.
- ~~**Por campo:** Implementado mediante reflexión (reflection). No recomendado por dificultar las pruebas unitarias, ocultar dependencias y aumentar el acoplamiento.~~




```
public class MovieLister {

	private MovieFinder finder;

	// Inyección por constructor
	public MovieLister(MovieFinder finder) {
		this.finder = finder;
	}

	// Inyección por setter
	public void setMovieFinder(MovieFinder finder) {
		this.finder = finder;
	}

	..
}
```

Nota: Cuando existe más de un constructor en la clase MovieLister, es necesario incluir la anotación @Autowired en el constructor que Spring debe utilizar para la inyección de dependencias.

Configuración de Spring

Spring permite configurar los beans (objetos administrados) a través del contexto de aplicación, facilitando la implementación de la inyección de dependencias.

```
@Configuration
public class AppConfiguration {

    @Bean
    public MovieLister movieLister() {
        return new MovieLister(movieFinder());
    }

    @Bean
    public MovieFinder movieFinder() {
      // Aquí podemos cambiar de implementación
      return new MovieFinderImpl();
    }

}
```

Es posible simplificar toda esta configuración utilizando anotaciones. Para ello, basta con añadir **@Component** (o sus derivadas como @Service, @Repository, etc.) a las clases MovieLister y MovieFinderImpl, permitiendo que Spring las detecte automáticamente.

Cuando existen múltiples implementaciones de una interfaz como MovieFinder, es necesario resolver la ambigüedad mediante las anotaciones **@Primary** (para definir la implementación por defecto) o **@Qualifier** (para seleccionar una implementación específica).

Orientación a Aspectos (AOP)

Spring AOP permite modularizar funcionalidades transversales como logging, seguridad y transacciones, manteniéndolas separadas del código de negocio principal.

Casos de uso comunes

Transacciones

@Transactional aplica gestión de transacciones que abarquen diferentes operaciones sobre bases de datos.

Auditoría

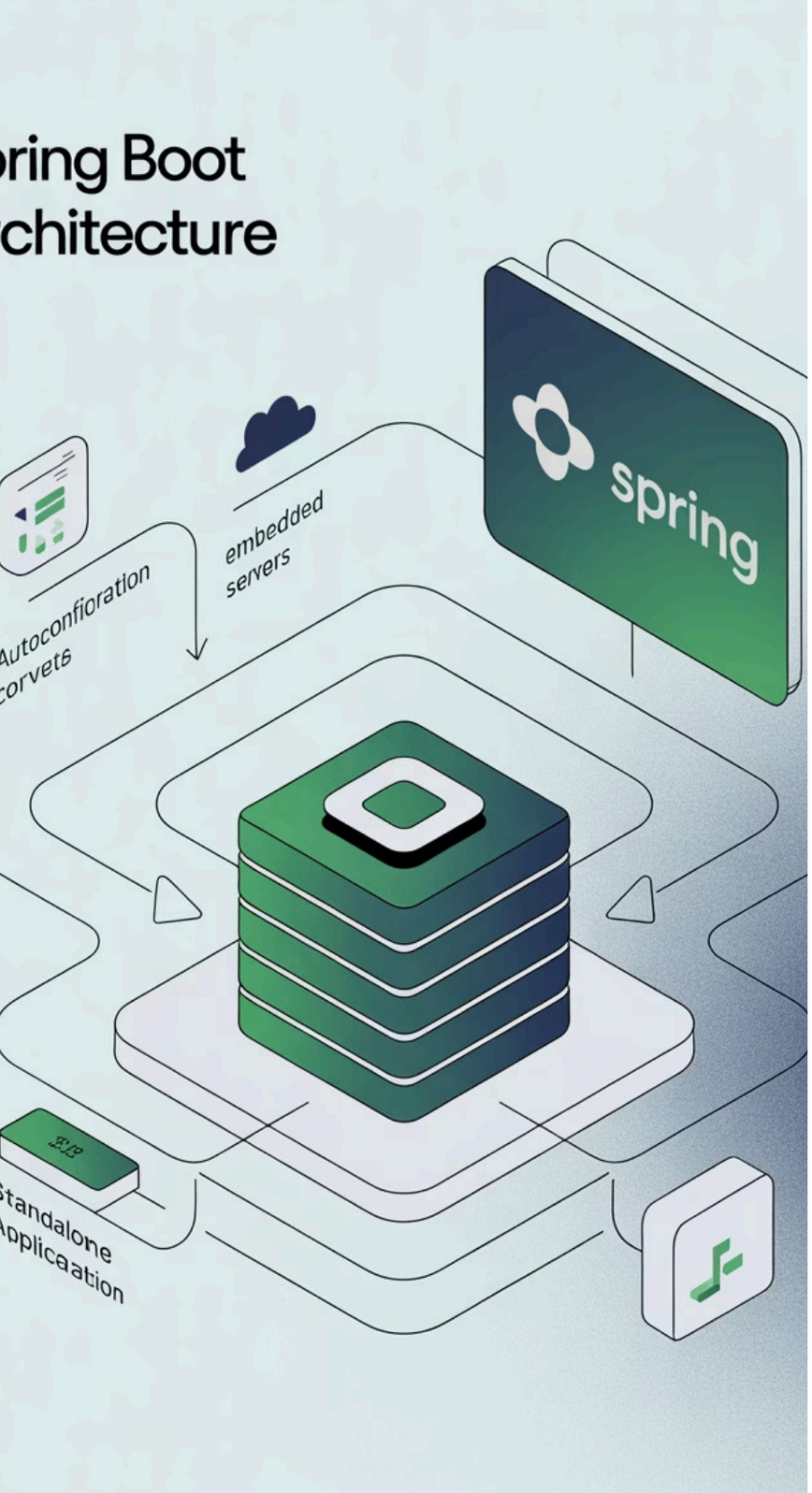
@CreatedDate, @LastModifiedDate interceptan llamadas para registrar quién, cuándo y qué operaciones se realizan.

Seguridad

@Secured verifica autorizaciones antes de permitir acceso a métodos sensibles.

Caché

@Cacheable y @CacheEvict para gestión automática de cachés de datos.



Spring Boot

Simplifica la complejidad de Spring

Spring Boot es un *"agilizador"* de Spring que simplifica la creación de aplicaciones Java **standalone** (autónomas), reduciendo la cantidad de código boilerplate y XML.

Configuración automática

Mediante la *autoconfiguration* se elimina gran parte de la configuración manual necesaria, detectando y configurando dependencias en el classpath

Servidores embebidos

Incluye servidores como Tomcat o Jetty embebidos, listos para ejecutar

Aplicaciones independientes

Permite crear aplicaciones auto-arrancables (java -jar) que se ejecutan sin necesidad de servidores externos

Spring Boot: starters

spring-boot-starter-web

Incluye Spring MVC, un servidor Tomcat embebido y Jackson (JSON) para el desarrollo de aplicaciones web RESTful

spring-boot-starter-data-jpa

Para acceso a bases de datos relacionales. Contiene hibernate como implementación de referencia de JPA

spring-boot-starter-security

Para añadir autenticación y autorización y protección contra vulnerabilidades comunes

spring-boot-starter-actuator

Expone endpoints que proporcionan información sobre la aplicación en producción para su monitorización

spring-boot-starter-test

Reúne JUnit, Spring Test, Mockito, AssertJ y Hamcrest para la implementación de pruebas

spring-boot-starter-logging

Starter para generación de logs en Spring Boot (SLF4J + Logback).

spring-boot-starter-mail

Facilita el envío de correos vía SMTP usando JavaMail.

spring-boot-starter-validation

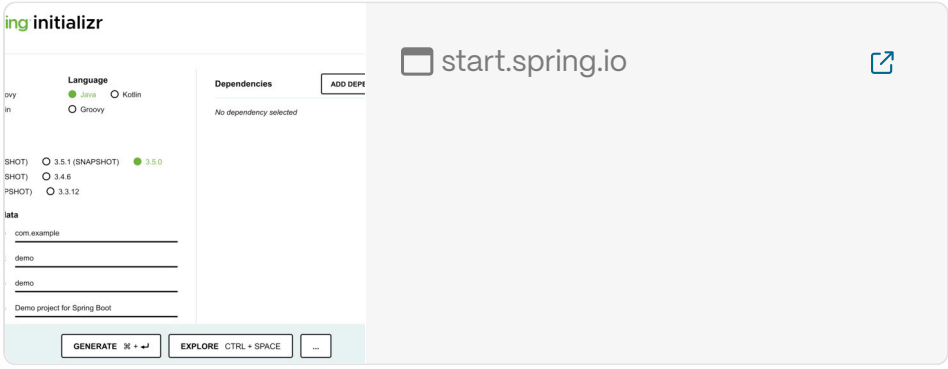
Integra Bean Validation (JSR 380) mediante Hibernate Validator.

Spring Initializr

Se suele usar *Spring Initializr* (**https://start.spring.io**) para generar un proyecto Maven preconfigurado

Este proporciona:

- Un POM con el *parent* de Spring Boot
- Dependencias *starter* (con versiones predeterminadas para bibliotecas comunes) según módulos seleccionados
- Estructura de proyecto estándar



Clase Principal de Spring Boot

@SpringBootApplication

La clase principal se anota con esta anotación, la cual incluye a otras como @Configuration, @EnableAutoConfiguration, @ComponentScan

Método main

Inicializa el contexto de aplicación, configura los beans necesarios (*@Configuration*) y arranca el servidor web embebido y ejecuta cualquier CommandLineRunner.

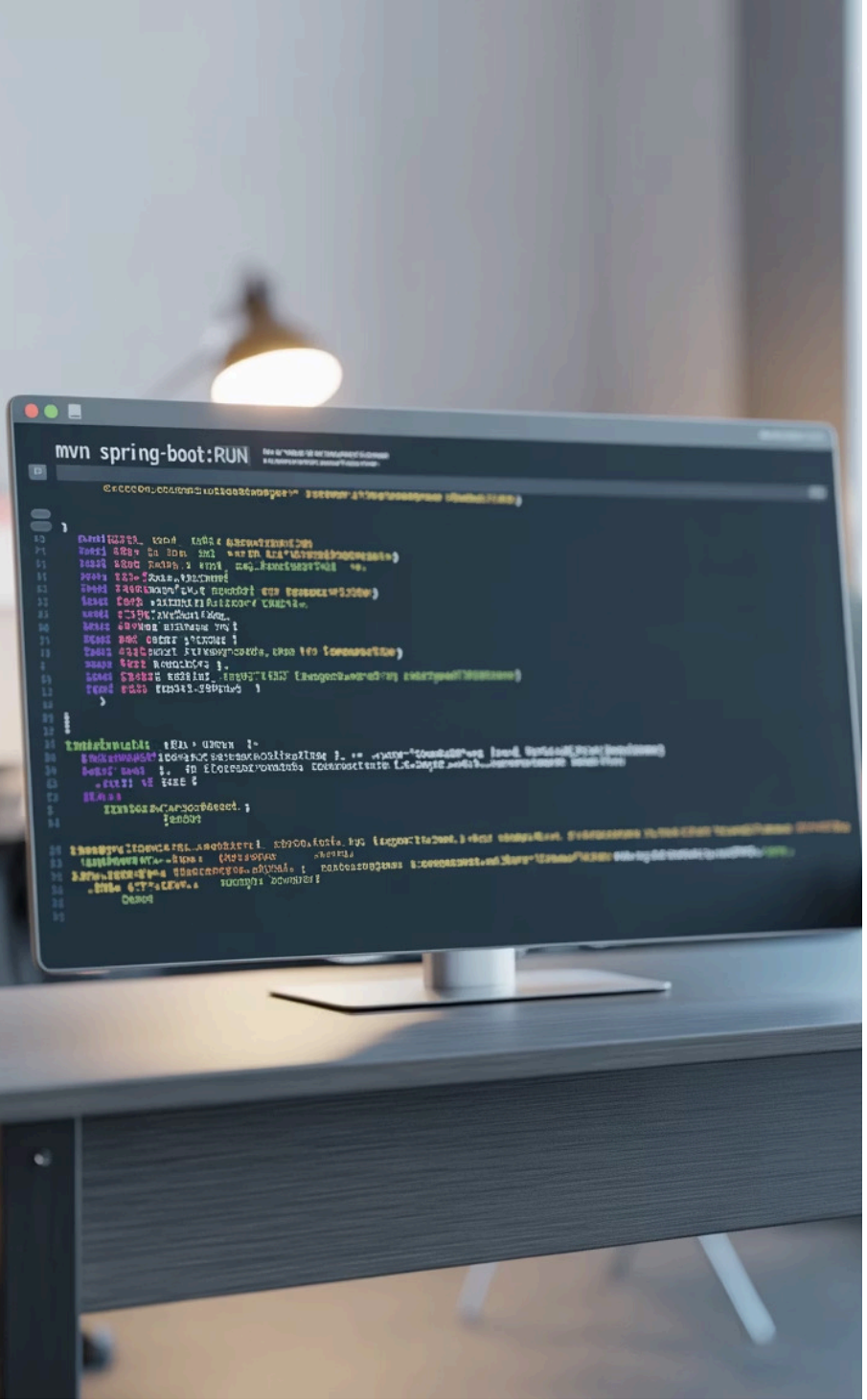
Escaneo de componentes

Spring Boot escanea (*@ComponentScan*) automáticamente el paquete de la clase principal y subpaquetes en busca de anotaciones ciertas predefinidas: @Component, @Controller, @Service, @Repository, etc.

Autoconfiguración

Configura (*@EnableAutoConfiguration*) automáticamente beans basándose en las dependencias del classpath. Por ejemplo, si H2 está en el classpath, Spring Boot configura automáticamente una base de datos en memoria.

```
package com.example.demo;

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

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {

        SpringApplication.run(DemoApplication.class, args);
    }

}
```



Spring Boot y Maven

Plugin de Spring Boot

Spring Boot incluye un plugin de Maven (spring-boot-maven-plugin) para facilitar el ciclo de vida

Ejecución directa

Mediante mvn **spring-boot:run** se compila y se levanta la aplicación

Empaquetado ejecutable

También crea ejecutables *fat JAR* con dependencias incluidas usando **mvn package**

Hot reload

Incluyendo las spring-boot-devtools, cada vez que se detectan cambios en el código se reinicia el servidor. Incluye soporte para LiveReload en el navegador

Hello World!

```
package es.uca.cursojava.springintro;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

	@GetMapping("/")
	public String hello() {
		return "Saludos desde la Universidad de Cádiz!";
	}

}
```

Iniciando...

```
/Users/ivanruizrube/.sdkman/candidates/java/17.0.7-tem/bin/java ...
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.0)
```

```
com.example.demo.DemoApplication          : Starting DemoApplication using Java 17.0.7 with PID 3415 (/Users/ivanruizrube/Downloads/demo/build/
com.example.demo.DemoApplication          : No active profile set, falling back to 1 default profile: "default"
o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat initialized with port 8080 (http)
o.apache.catalina.core.StandardService    : Starting service [Tomcat]
o.apache.catalina.core.StandardEngine     : Starting Servlet engine: [Apache Tomcat/10.1.41]
o.a.c.c.C.[Tomcat].[localhost].[/]        : Initializing Spring embedded WebApplicationContext
w.s.c.ServletWebServerApplicationContext  : Root WebApplicationContext: initialization completed in 351 ms
o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port 8080 (http) with context path '/'
com.example.demo.DemoApplication          : Started DemoApplication in 0.748 seconds (process running for 1.201)
```

Resumen

Spring Framework proporciona un ecosistema robusto y flexible para el desarrollo profesional de aplicaciones Java empresariales.

Se fundamenta en paradigmas de diseño avanzados como Inversión de Control (IoC), Inyección de Dependencias (DI) y Programación Orientada a Aspectos (AOP), facilitando la creación de código altamente modular, testeable y mantenible.

Framework Core

Contenedor ligero que implementa patrones IoC y DI para gestionar eficientemente el ciclo de vida de los beans y sus dependencias.

Spring Boot

Plataforma de desarrollo que simplifica la creación de aplicaciones mediante autoconfiguración inteligente, servidores embebidos y despliegue sin dependencias externas.

Ecosistema

Amplia gama de módulos especializados e integrables que cubren seguridad, acceso a datos, desarrollo web, microservicios, integración y computación reactiva.

Herramientas

Suite completa de utilidades que incluye Spring Initializr para generación de proyectos, plugins de Maven/Gradle, y herramientas de desarrollo que optimizan el flujo de trabajo.