Spring Framework

Que es Spring Framework

Es un robusto Framework para el Desarrollo de Aplicaciones Empresariales en el lenguaje Java

- Aplicaciones Web MVC
- Aplicaciones empresariales
- Aplicaciones de escritorio
- Aplicaciones Batch
- Integración con REST/SOA
- Spring Data
- Spring Security

Características - CDI

- Gestión de configuración basada en componentes JavaBeans y aplica el principio Inversión de control, específicamente utilizando la inyección de dependencias (DI) para manejar relaciones entre los objetos, evitando relaciones manuales y creaciones de instancias explícitas con operador new, esto hace un bajo acoplamiento y alta cohesión, mejorando la reutilización y mantención de los componentes
- Spring en su CORE está basado en un contenedor liviano y es usado globalmente dentro de nuestra aplicación

Características - ORM y Persistencia

- Alta abstracción por sobre el API JDBC
- Integración con frameworks de persistencia como Hibernate, JPA, entre otros.
- Soporte de la lógica de negocio, específicamente en clases de acceso a datos (DAO Support)
- Componentes encargados de la gestión de transacciones de base de datos

Características - MVC

- La arquitectura **MVC** es uno de los principales componentes y tecnologías, y como su propio nombre nos indica implementa una arquitectura Modelo - Vista – Controlador
- Soporta varias tecnologías para generación de las vistas, entre ellas JSP, Thymeleaf, FreeMarker, Velocity, Tiles, iText, y POI (Java API para archivos Microsoft Office)

Características - AOP

- AOP es un paradigma de programación que permite modularizar las aplicaciones y mejorar la separación de responsabilidades entre componentes y/o clases.
    - Similar a los componentes de Inyección de Dependencia, AOP tiene como objetivo mejorar la modularidad de nuestra aplicación.
- Spring amplia la programación orientada a aspectos (AOP) para incluir servicios tales como manejo de transacciones, seguridad, logger etc.

¿Por qué usar Spring Framework?

- **Modularidad de Componentes** a través del patrón Inyección de Dependencia (CDI)
  - Promueve la composición y modularidad entre las partes que componen una aplicación
  - Plain Old Java Objects mantienen su código limpio, simple y modular, bajo acoplamiento y alta cohesión

¿Por qué usar Spring Framework?

- Simplicidad
  - Las aplicaciones con Spring son simples y requieren mucho menos código (Java y XML) para la misma funcionalidad
- Capacidad de pruebas unitarias
  - Dependencias limpias, actualizadas y los justo y necesaria, aseguran que la integración con unit testing sea muy simple
  - Clases POJO se pueden testear sin estar atado al framework

¿Por qué usar Spring Framework?

- Facilidad de configuración
  - Se elimina la mayor parte del código repetitivo y la configuración de XML a partir de sus aplicaciones y mayor uso de anotaciones
- AOP (Aspect Oriented Programming)
  - Programación declarativa AOP, paradigma de programación que permite modularizar las aplicaciones y mejorar la separación de responsabilidades entre módulos y/o clases aspectos
  - Facilidad de configurar aspectos, soporte de transacciones, seguridad.

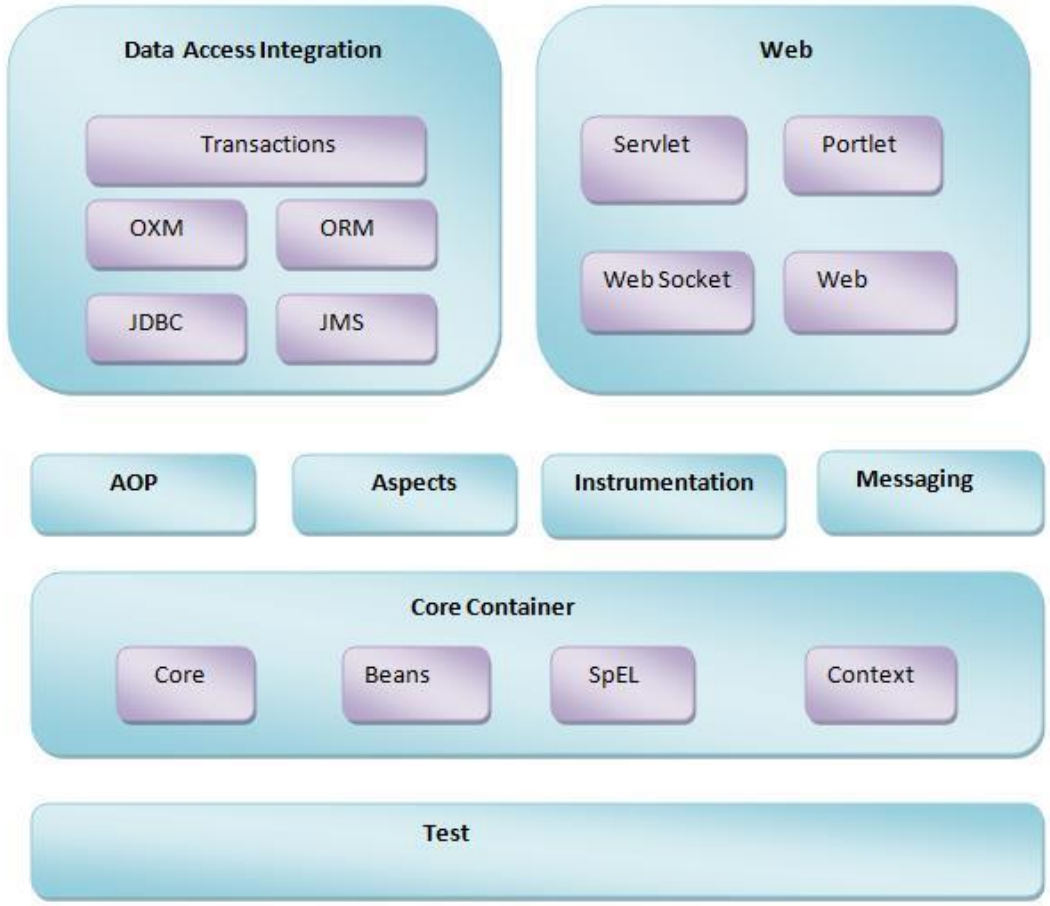
¿Por qué usar Spring Framework?

- Diseño orientado a interfaces
  - Programación basadas en contratos de implementación, permitiendo al usuario centrarse en la funcionalidad, ocultando el detalle de implementación
- Plenamente probado, seguro y confiable
  - Spring ha sido probado y utilizado en diversos proyectos alrededor del mundo, como en Instituciones Bancarias, Aseguradoras, Instituciones Educativas y de Gobierno, entre muchos otros tipos de proyectos y empresas

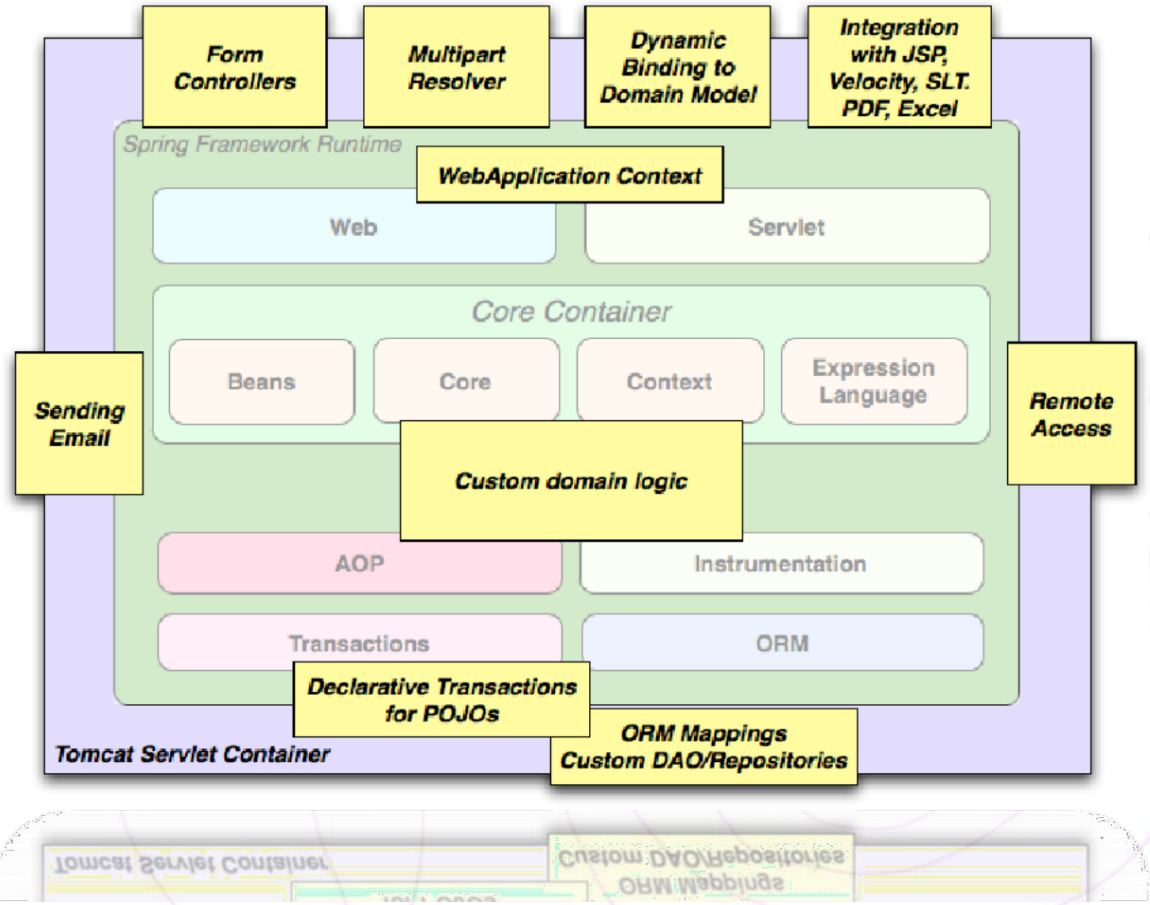
¿Por qué usar Spring Framework?

- Productividad
  - Ganancias de productividad y una reducción en el tiempo de desarrollo e implementación utilizando Spring
- Integración con otras Tecnologías
  - EJB 3.2 (Lógica de negocio)
  - JPA, Hibernate, iBates, JDBC (Pesistencia)
  - Velocity, etc (Vista)
  - JSF2, Struts, etc (Capa web)

Arquitectura Spring Framework



Arquitectura Spring Framework



Arquitectura Spring Framework

La arquitectura se compone en distintas capas, cada una tiene su función específica:

- **Capa Web:** Spring simplifica el desarrollo de interfaces de usuario en aplicaciones Web MVC mediante el Soporte de varias tecnologías para generación de contenido, entre ellas JSP, Thymeleaf, FreeMarker, Velocity, Tiles etc.

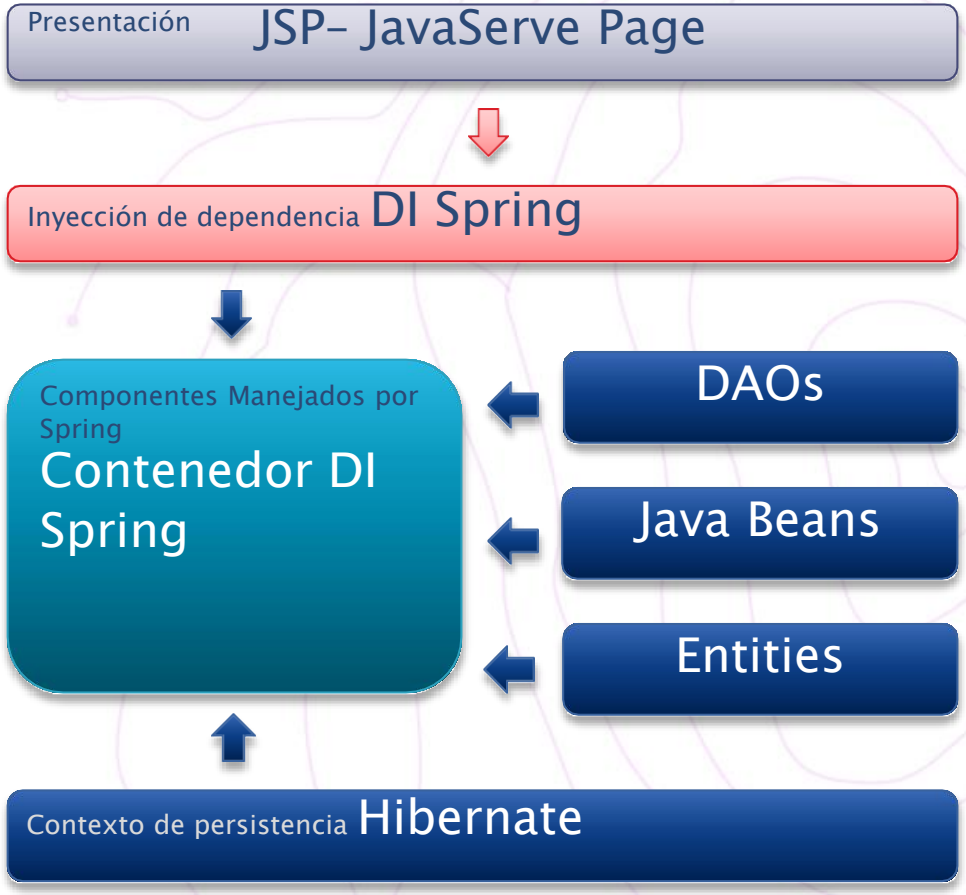
Arquitectura Spring Framework

- **Capa Lógica de Negocio:** en esta capa podemos encontrar tecnología como los Java Beans (POJOs), Dao Support, Services, EJBs etc y clases Entities.
- **Capa de Datos:** aquí vamos a encontrar tecnologías JDBC, ORM (JPA, Hibernate, etc), Datasource y conexiones a bases de datos.

¿Qué es la Inyección de Dependencia?

- Inyectar es justamente suministrar a un objeto una referencia de otros que necesite según la relación, tiene que plasmarse mediante configuración XML o la anotación @Autowired.

¿Qué es la Inyección de Dependencia?



¿Qué es la Inyección de Dependencia?

También es un tipo de Inversión de Control (IoC):

- “CDI Container" Maneja los contextos y resuelve dependencias de componentes mediante la asociación e inyección de objetos (push)
- En contra-oposición de la creación explícita (operador new) de objetos (pull)

¿Qué es la Inyección de Dependencia?

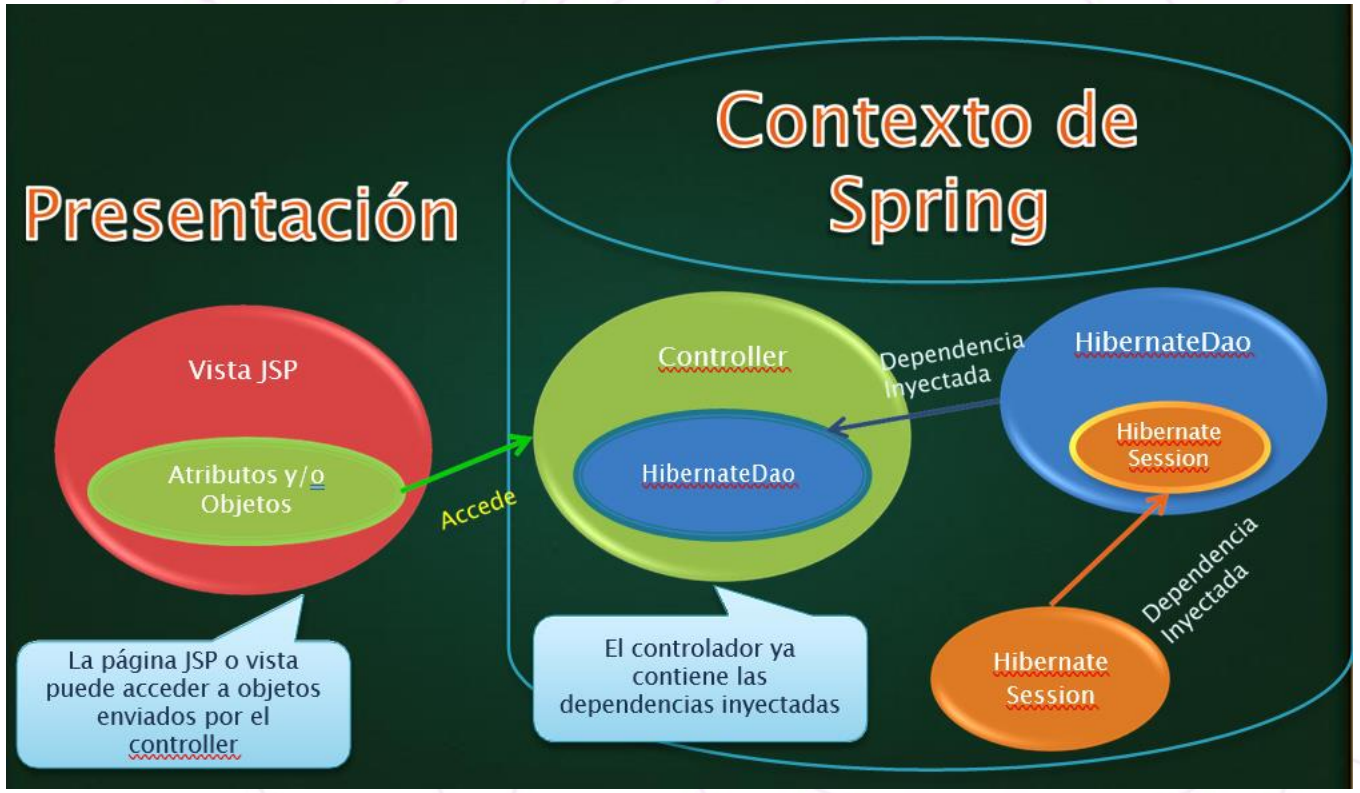
- El "Contenedor" se encarga de gestionar las instancias y relaciones (así como sus creaciones y destrucciones) de los objetos
- El objetivo es lograr un bajo acoplamiento entre los objetos de nuestra aplicación
- Martin Fowler lo llama inyección de dependencias (DI)

¿Qué es la Inyección de Dependencia?

El mecanismo de inyección (injection) permite a un componente A obtener de un contexto una referencia de una instancia de un componente B

Haciendo que el contenedor de aplicaciones ”inyecte” el componente B en una variable del componente A, en tiempo de ejecución

¿Qué es la Inyección de Dependencia?



Variantes para la inyección de dependencias

Inyección de dependencia vía Constructor

- Las dependencias se proporcionan a través de los constructores de una clase o componente,
- Pasándose como argumento

Mediante método Setter

- Las dependencias se establecen mediante métodos setter de un componente (estilo JavaBean)

Mediante atributo

- Las dependencias se establecen directamente sobre el atributo de la clase componente o beans
- Es la forma más típica y recomendada

Inyección de Dependencia vía atributos

- En general, las dependencias se establecen a través de los atributos de un componente Spring usando la anotación @Autowired

```
public class InyeccionSetter {
    @Autowired
    private Dependencia miDependencia;

}
```

Inyección de Dependencia vía Setter

- Las dependencias se establecen a través de los métodos setter de un componente Spring usando la anotación @Autowired

```
public class InyeccionSetter {
    private Dependencia miDependencia;

    @Autowired
    public void setMiDependencia(Dependencia dep) {
        this.miDependencia = dep;
    }
}
```

Inyección de Dependencia vía Constructor

```java
public class InyeccionConstructor {
    private Dependencia miDependencia;

    @Autowired
    public InyeccionConstructor(Dependencia dep) {
        this.miDependencia = dep;
    }
}
```

Beans

- El término “bean" (o componente) se utiliza para referirse a cualquier componente manejado por Spring
- Los “bean" son clases en forma de JavaBeans
  - Sin *args* en el constructor.
  - Métodos *getter* y *setter* para los atributos
- Atributos de los “beans" pueden ser valores simples o probablemente referencias a otros “beans“
- Los "beans" pueden tener varios nombres

Anotación @Autowired

- Se utiliza en el código java, en clases sobre atributos, métodos, setter, constructor para especificar requerimiento DI (en vez de archivo XML)
- Ejemplo @Autowired Clase Target

```
public class Persona {
    private String nombre = "Andrés Guzmán";
    private int edad = 35;
    private float altura = 1.78;
    private boolean esProgramador = true;
    @Autowired
    private Direccion direccion;

    public Direccion getDireccion() {
        return direccion;
    }
}
```

Auto-scanning

- Puede ser usado para crear instancias de los objetos beans en lugar de declararlos en clases de configuración (anotación @Configuration)
- Spring Boot lo resuelve de forma automática
- Los beans deben ser anotado con la anotación **@Component**
  - Cualquier beans anotado con **@Component** bajo el package base serán instanciados y manejados por el contenedor DI de Spring

Bean anotado con @Component

```java
package com.misiontic2022.dominio;

import org.springframework.stereotype.Component;

@Component
public class Direccion {
    private int numeroCalle = 1234;
    private String nombreCalle = "Av. Kennedy";
    private String ciudad = "Pereira";
    private String pais = "Colombia";

    public int getNumeroCalle() {
        return numeroCalle;
    }
    // ... etc ...
}
```