# APPUNTI JAVA SPRING E HIBERNATE

# Indice generale

# COSA SONO SPRING E SPRING BOOT

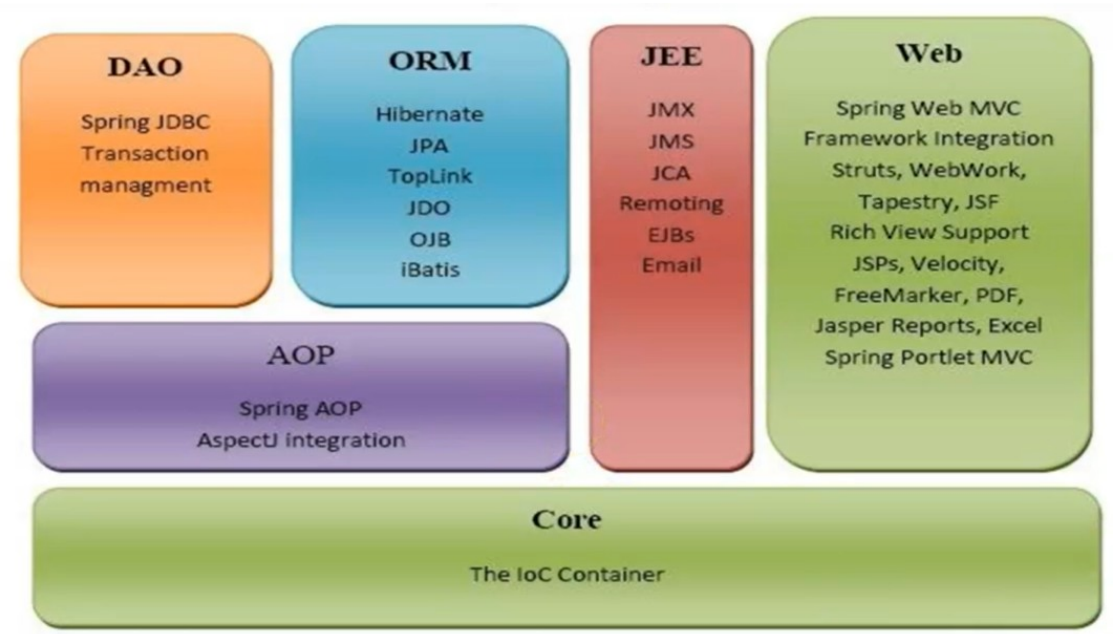
Spring è un framework di JAVA.

Dalla versione 5.0.2 diventa il framework più popolare per lo sviluppo di applicazioni JAVA EE. Infatti, quando si sente parlare di framework di JAVA, si sente sempre e solo parlare di Spring.

Spring è un framework con una struttura modulare, dove una parte dei moduli sono facoltativi, ma uno in particolare è obbligatorio. I moduli possiamo utilizzarli in parte, o tutti, senza stravolgere l'architettura del progetto.

Qui di seguito possiamo vedere la rappresentazione dei moduli di Spring:

Il

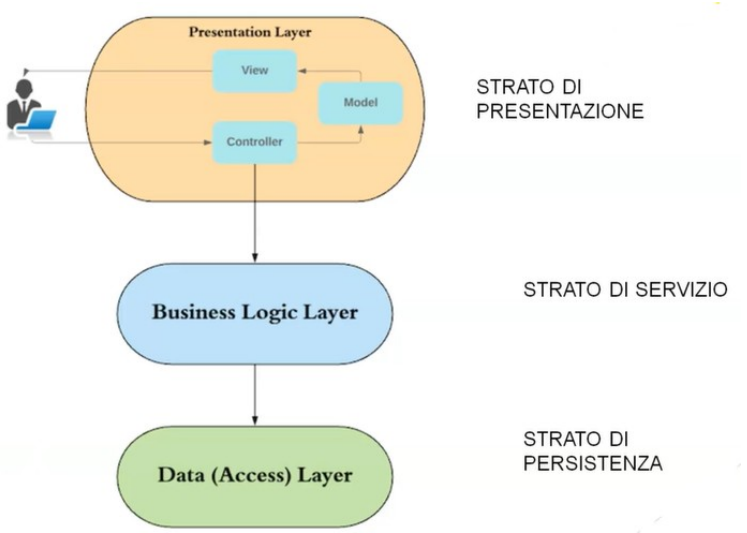



modulo obbligatorio di cui parlavamo è il Core, che è il motore del framework. Gli altri moduli possono essere utilizzati facoltativamente.

# DESIGN PATTERN

Un design pattern è uno schema architetturale da seguire che semplifica lo sviluppo di applicazioni enterprise. In pratica è una linea guida per scrivere codice pulito, manutenibile e riutilizzabile.

L'MVC (Model View Controller) è un design pattern che ha come scopo la divisione del modello, della vista e del controller.




Quando si sviluppa un'applicazione web, si hanno 3 strati:

- strato di presentazione o presentation layer;
- strato di servizio o business logic layer;
- strato di persistenza o data access layer;

Quando un utente manda una richiesta, essa viene intercettata da un controller. Il controller comunica con lo strato di servizio, al cui interno sono presenti tutte le logiche di business. I dati che elaborerà lo strato di servizio, verranno presi dallo strato di persistenza, che a sua volta comunicherà con un database. Quindi, all'interno dello strato di persistenza verranno effettuate delle query per interagire con il database. La risposta verrà recuperata dallo strato di persistenza, dopodichè verrà richiamata dallo strato di servizio per elaborarla. Il risultato di tale elaborazione viene restituito al controller che, a sua volta, lo passerà alla view, in maniera tale che l'utente possa visualizzare la risposta.

# SPRING BOOT

Sebbene Spring offra numerosi vantaggi allo sviluppatore, uno dei suoi punti negativi è la sua difficoltà di configurazione. Spring Boot nasce con lo scopo di risolvere questo problema.

Spring Boot è un progetto Spring che ne semplifica la configurazione e ne permette l'esecuzione senza server, avendone già uno integrato.

# CONFIGURAZIONE DI SPRING

La configurazione di Spring può avvenire in 3 modi:

- file .xml;
- classi di configurazione JAVA;
- annotation (che utilizzeremo in questo corso).

Vediamo una lista di annotation più utilizzate in Spring:

- @Component
- @Controller (@RestController)
- @Service
- @Repository
- @RequestParam
- @RequestMapping
- @PathVariable
- @RequestBody
- @Autowired
- @ComponentScan
- @Bean
- @Configuration
- @Size
- @Valid
- @EnableAutoConfiguration
- @SpringBootApplication
-

# IOC (INVERSION OF CONTROL) E DI (DEPENDENCY INJECTION)

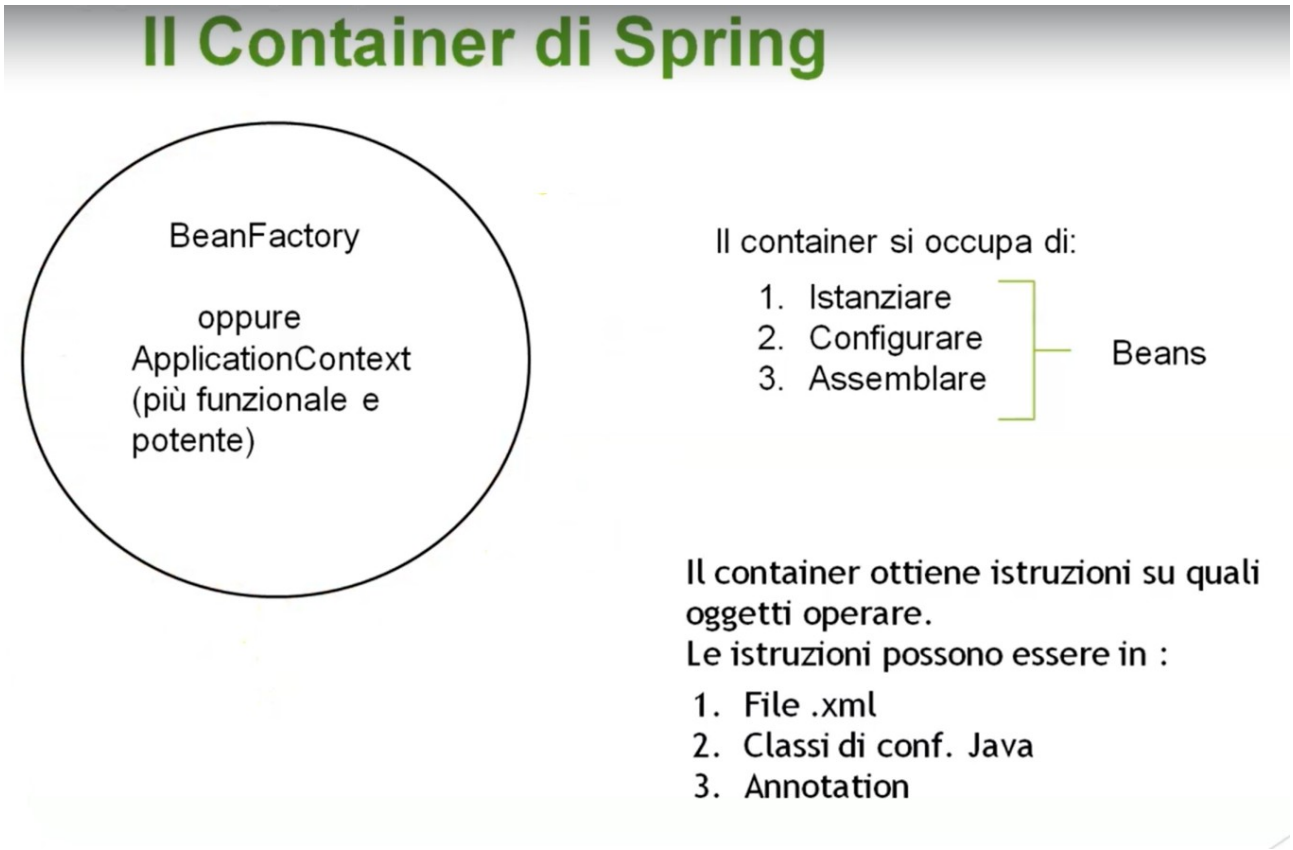
I due concetti che andremo a vedere di seguito sono fondamentali per il funzionamento di Spring.

L'inversion of control è un principio architetturale (o design pattern) basato sull'inversione del flusso del sistema. Come già spiegato, un design pattern è uno schema da seguire per permettere un più facile sviluppo, manutenibilità e riusabilità del codice.

Attraverso l'inversion of control, non è più il programmatore a doversi occupare di creare e istanziare oggetti, o invocare metodi, ma lo farà Spring attraverso una certa configurazione.

La dependency injection è un'implementazione dell'inversion of control e permette di iniettare le dipendenze di una classe all'esterno, senza che tali dipendenze vengano inizializzate dalla classe stessa.

Partiamo con un esempio teorico:

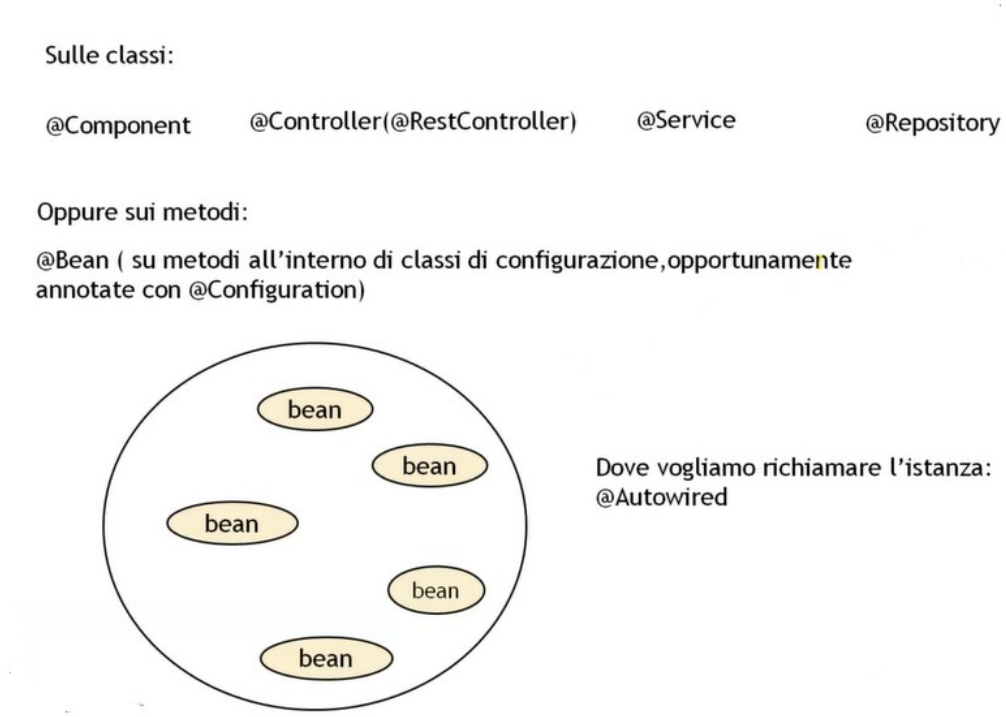



Spring ha un container che sta dietro le quinte. Il container è un'istanza di BeanFactory, oppure di ApplicationContext. Tale container, con il fatto che applica la dependency injection, si occupa di istanziare, configurare e assemblare i Beans. Il container deve essere però configurato in uno dei 3 modi elencati quando abbiamo parlato delle configurazioni di Spring, ossia:

- file .xml;
- classi di configurazione JAVA;
- annotation.

Sulle classi, ogni qualvolta mettiamo un annotation tra @Component, @Controller(@RestController), @Service, o @Repository, Spring le trasforma in un bean e lo mette all'interno del container. Quando poi ci serve l'istanza di quella classe, utilizziamo l'annotation @Autowired per andare a pescare il bean messo precedentemente nel container.

La stessa operazione può essere fatta mettendo l'annotation @Bean sopra alcuni metodi che si trovano nelle classi di configurazione, opportunamente annotate con @Configuration.




Vediamo dal punto di vista pratico questo concetto:

Persona.java

```
package com.example.dependency_injection;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/* Quando inizializziamo il progetto, Spring va a vedere tutte le classi
 * con l'annotation @Component, ne crea un bean e lo inserisce nel suo
 * cantainer
 */
@Component
public class Persona {
    String nome;
    String cognome;

    @Autowired
    Indirizzo indirizzo;

}
```

Indirizzo.java

```java
package com.example.dependency_injection;

import org.springframework.stereotype.Component;

@Component
public class Indirizzo {
    String via;
    Integer civico;

}
```

MainDependencyInjection.java

```java
package com.example.dependency_injection;

import org.springframework.beans.factory.annotation.Autowired;

public class MainDependencyInjection {

    /* L'annotation @Autowired va sopra la dichiarazione
     * dell'istanza di una classe annotata precedentemente
     * con l'annotation @Component. In questo modo non
     * abbiamo bisogno di inizializzare l'istanza con new,
     * dato che lo fa già Spring dietro le quinte/
     */
    @Autowired
    static Persona p;

    public static void main(String[] args) {
        p.nome = "";
        p.cognome = "";
    }

}
```

# ALTRO ESEMPIO DI DEPENDENCY INJECTION