**Spring Framework**

* Spring Framework là một Framework mã nguồn mở, dùng để phát triển các ứng dụng Java.
* Mục tiêu chính của Spring là giúp phát triển các ứng dụng J2EE một cách dễ dàng hơn dựa trên mô hình sử dụng POJO (Plain Old Java Object)
* Spring được chia làm nhiều module khác nhau, tùy theo mục đích phát triển ứng dụng mà ta dùng 1 trong các module đó



+ **Test**: Tầng này cung cấp khả năng hỗ trợ kiểm thử (Unit test code)

+ **Spring Core Container:**

* Spring core, bean cung cấp tính năng IOC và Dependency Injection
* Spring Context hỗ trợ đa ngôn ngữ (internationalization), các tính năng Java EE như EJB, JMX
* Expression Language được mở rộng từ Expresion Language trong JSP. Nó cung cấp hỗ trợ việc setting/getting giá trị, các method cải tiến cho phép truy cập collections, index, các toán tử logic…

+ **AOP, Aspects and Instrumentation:** Những module này hỗ trợ cài đặt lập trình hướng khía cạnh (Aspect Oriented Programming), hỗ trợ tích hợp với AspectJ.

+ **Data Access / Integration:** Nhóm này bao gồm JDBC, ORM, OXM, JMS và module Transaction. Những module này cung cấp khả năng giao tiếp với database

+ **Web:** Hay còn gọi là Spring MVC Nhóm này gồm Web, Web-Servlet… hỗ trợ việc tạo ứng dụng web.

* Một số các Project khác của Spring: <https://spring.io/projects>

+ Spring MVC

+ Spring Security

+ Spring Boot

+ Spring Batch

+ Spring Social

+ Spring IO

+ Spring Cloud

+ Spring Mobile

+ Spring for Android

+ Spring Session

**Spring Core**

1. **IoC container**

* **IoC (Inversion of Control) container** được xem là cốt lõi của Spring Framework, giúp lập trình viên quản lý quá trình thực thi **DI (Dependency Injection)** trong ứng dụng một cách tự động.
* Spring IoC Container sẽ tạo các đối tượng, liên kết chúng lại với nhau, cấu hình các đối tượng và quản lý vòng đời của chúng từ lúc tạo ra cho đến lúc bị hủy. Các đối tượng tạo nên xương sống của ứng dụng và được quản lý bởi Spring IoC Container được gọi là các **bean**.
* Hai package **org.springframework.**beans và **org.springframework.context** là những nền tảng của Spring IoC container
* Interface **BeanFactory** của **package org.springframework.beans** cung cấp framework cấu hình và chức năng cơ bản.
* Interface **ApplicationContext** của **package org.springframework.context:** ApplicationContext được xây dựng BeanFactory nhưng nó có thêm một số chức năng mở rộng như tích hợp với Spring AOP, xử lý message, context cho web application. Chúng ta có thể xem ApplicationContext giống như là một sub interface bổ trợ cho BeanFactory. Do đó trong thực tế, chúng ta sẽ ưu tiên sử dụng ApplicationContext hơn là BeanFactory.

[[](https://docs.spring.io/spring/docs/current/spring-framework-reference/images/container-magic.png)](https://docs.spring.io/spring/docs/current/spring-framework-reference/images/container-magic.png)

Tổng quan luồng hoạt động của Spring IoC Container. Spring IoC Container (mà đại diện là ApplicationContext) sau khi nạp các POJO, sẽ kết hợp với configuration metadata (được biểu diễn bằng XML hoặc Java code) để khởi tạo, cấu hình và lắp ráp các bean. Lúc này, chúng ta sẽ có một ứng dụng được cấu hình đầy đủ và có thể thực thi được.

1. **Dependency Injection (DI)**

* Dependency Injection (có thể dịch tiêm các thành phần phụ thuộc) là một sức mạnh nổi bật của Spring Framework.
* Dependency Injection là một design pattern implement theo nguyên lý Inversion of Control (IoC). DI truyền vào sự phụ thuộc giữa các đối tượng. Thay vì để đối tượng A trực tiếp khởi tạo các đối tượng B, C, D, ... thì ta sẽ tiêm (inject) các đối tượng phụ thuộc của B, C, D vào A
* Ta sẽ có một ví dụ đơn giản về Dependency Injection như sau: Bạn có một web controller có nhiệm vụ lưu thông tin gửi từ form của người dùng. Theo nguyên lý Đơn trách nhiệm nói trên, bạn không muốn lớp controller tương tác với cơ sở dữ liệu. Thay vào đó, bạn sẽ sử dụng một lớp service để làm công việc này. Như vậy, controller của bạn sẽ chỉ phải xử lý dữ liệu của form (get form data, validate data, ...) rồi gọi một phưong thức của lớp service được inject để lưu dữ liệu. Controller không cần phải quan tâm hay lo lắng về kết nối cơ sở dữ liệu, pooling hay bảng nào sẽ được update. Cũng như service không cần phải biết request có những thông tin gì.
* Spring Framework hỗ trợ 2 cách thực hiện Dependency Injection (DI):

+ DI thông qua constructor

+ DI thông qua setter

* **Chúng ta nên thực hiện DI thông qua constructor hay setter?**

Chúng ta hoàn toàn có thể sử dụng kết hợp cả 2 cách. Nhưng theo khuyến nghị của Spring, chúng ta nên:

* DI thông qua constructor với các dependency bắt buộc phải có.
* DI thông qua setter với các dependency không bắt buộc phải có.
* Điều này có thể giúp DI trở nên linh động hơn. Tuy nhiên, có một số điều nên tránh cũng như hạn chế khi thực hiện DI là:

DI thông qua constructor: một constructor có thể có quá nhiều tham số => code smell => cần phải refactor code.

DI thông qua setter: dependency instance có thể bị null => cần phải check null trước khi thực hiện DI.