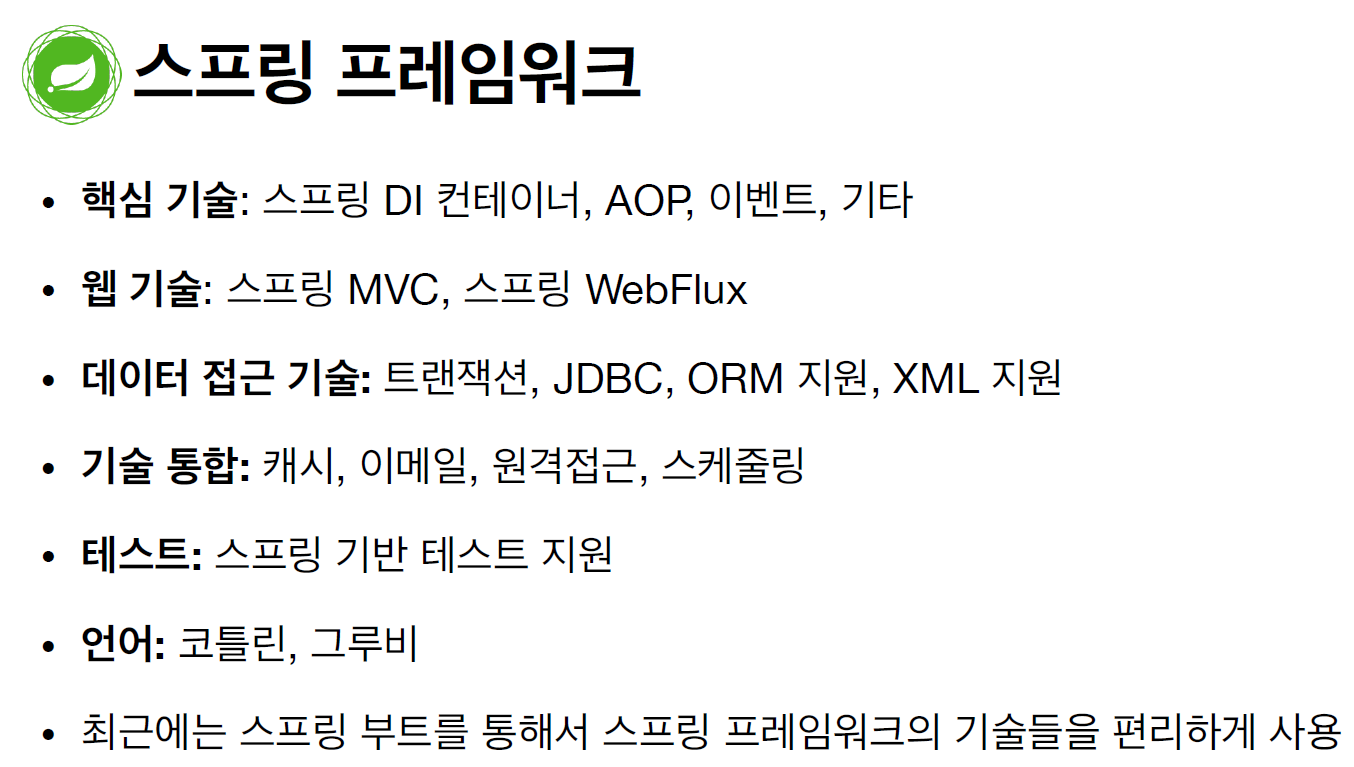
1. What is Spring Framework?

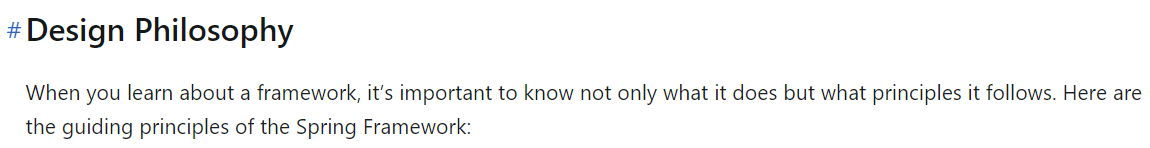


The Spring Framework is divided into modules.

At the heart are the modules of the **core container**, including a **configuration model** and a **dependency injection mechanism.**

Beyond that, the Spring Framework provides foundational support for different application architectures.

스프링 프레임워크의 디자인 철학



- Provide choice at every level

Spring lets you defer design decisions as late as possible. For example, you can switch persistence providers through configuration without changing your code. The same is true for many other infrastructure concerns ans integration with third-party APIs.

- Accomdate diverse perspectives.

Spring embraces flexibility and is not opinionated about how things should be done. It supports a wide range of application needs with different perspectives.

- Maintain strong backward compatibility.

Spring’s evolution has been carefully managed to force few breaking changes between versions. Spring supports a carefully chosen range of JDK versions and third-party libraries to facilitate maintenance of applications and libraries that depend on Spring.

- Care about API design.

The Spring team puts a lot of thought and time into making APIs that are intuitive and that hold up across many versions and many years.

-Set high standards for code quality.

The Spring Framework puts a strong emphasis on meaningful, current, and accurate javadoc. It is one of very few projects that can claim clean code structure with no circular dependencies between packages.

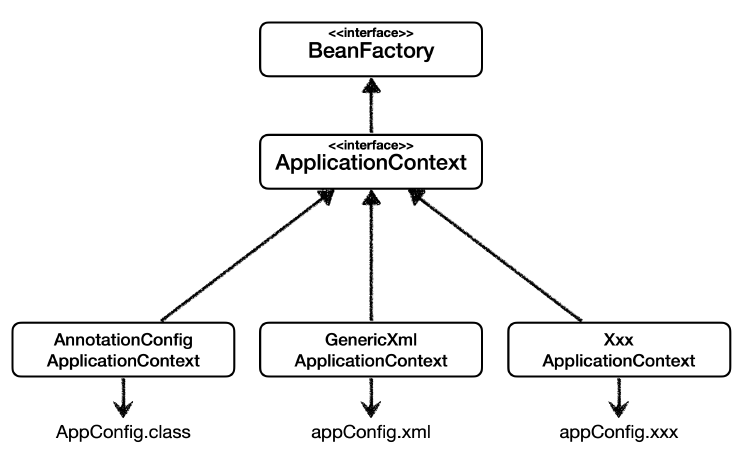
**Spring IoC (Inversion of Control) Conatiner**

의존성 주입(Dependency Injection)은 객체지향 프로그래밍에서 객체 간의 의존성을 효율적으로 관리하기 위해 사용되는 디자인 패턴이다. DI는 제어의 역전(IoC)의 한 형태로, 객체가 스스로 필요한 의존성을 직접 생성하거나 찾는 대신, 외부에서 주입받는 방식으로 동작한다. 객체의 의존성을 주입해주는 외부 컨테이너를 Ioc Container라고 한다.

객체지향의 다형성 만으로는 OCP(Open-Closed Principle) + DIP(Dependency Inversion Principle)를 지킬 수 없다. 스프링은 IoC Container를 통해 이를 지원한다. 순수하게 자바로 OCP, DIP 원칙들을 지키면서 개발을 해보면, 결국 DI 컨테이너를 만들게 된다.

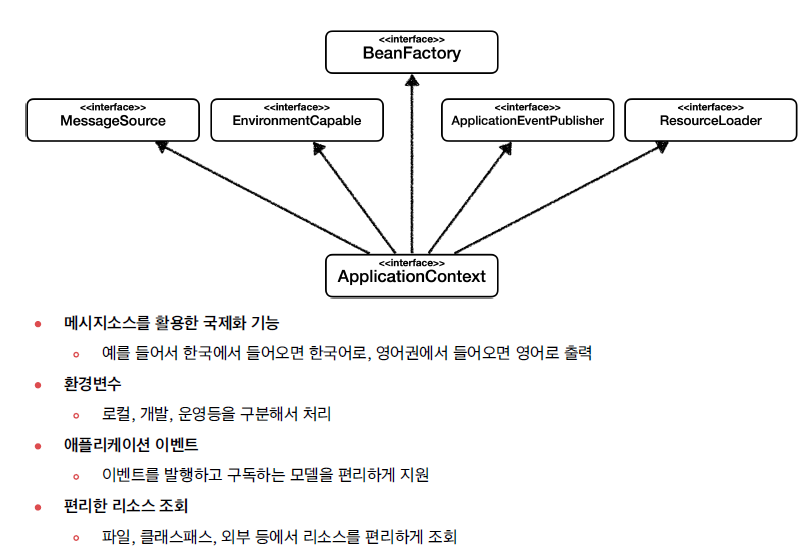
In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**. A bean is an object that is instantiated, assembled, and managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

**Spring IoC Container Interfaces.**

****

**BeanFactory** : provides the configuration framework and basic functionality.

**ApplicationContext** : Complete superset of the BeanFactory. It adds more enterprise-specific functionality.



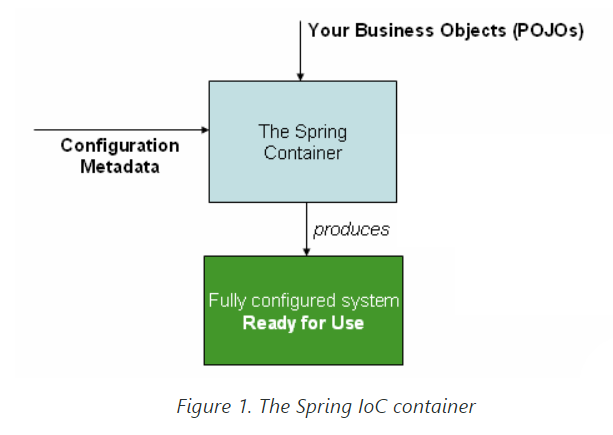
BeanFactory는 스프링 빈을 구성, 관리, 조회하는 기능을 제공하는 핵심적인 인터페이스이다.

ApplicationContext는 BeanFactory의 기능을 모두 상속받고, 운영환경에 적합한 부가기능들을 추가로 제공한다.

ApplicationContext interface represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the beans.

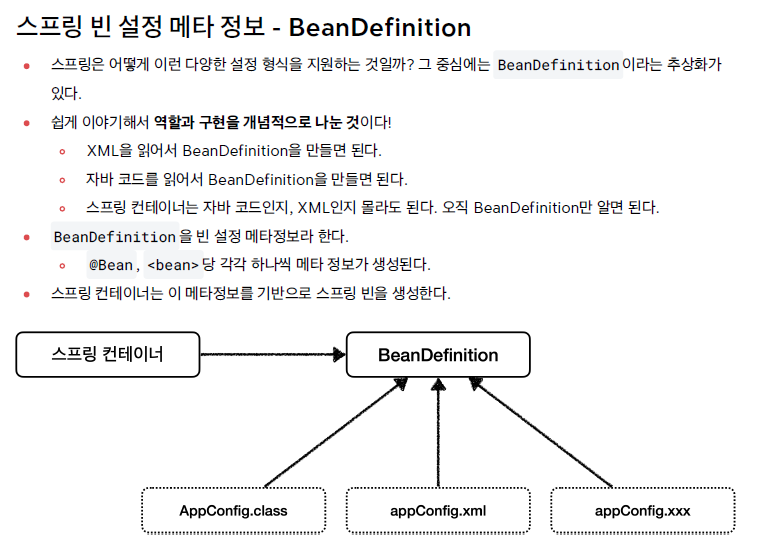
The container gets its instructions on the components to instantiate, configure, and assemble by reading **configuration metadata**. The configuration metadata can be represented as annotated component classes, …

IoC 컨테이너는 Configuration metadata를 읽어들이며 빈을 구성한다. 메타 데이터는 어노테이션 기반, XML 기반 등 다양한 형태로 구성이 가능하다.

****

As the preceding diagram shows, the Spring IoC container consumes a form of configuration metadata. This configuration metadata represents how you, as an application developer, tell the Spring container to instantiate, configure, and assemble the components in your application.

In most application scenarios, explicit user code is not required to instantiate one or more instances of a Spring IoC container. In a Spring Boot scenario, the application context is implicitly bootstrapped for you based on common setup conventions.



각각의 메타 데이터 형식이 BeanDefinition이라는 추상화를 만들어낸다. IoC 컨테이너는 이 BeanDefinition을 읽어들여 빈을 구성한다. 어떠한 메타데이터 형식이든 BeanDefinition만 구성이 가능하면 스프링 컨테이너를 동작시킬 수 있다.

