Spring Beans Guide

LAHIRU
LIYANAPATHIRANA

What is a Bean?

Spring Beans are the fundamental building blocks of the Spring Framework.

They are:

- Java objects
- Managed by the IoC Container
- Configured via XML or Annotations

The Spring IoC container is responsible for instantiating, configuring, and managing the beans.

Bean Scope

In the Spring Framework, a bean's scope determines its lifecycle, visibility, and instantiation within the Application Context.

Singleton – It is the default scope. It creates a single, shared instance per container, which is cached and reused for all requests.

Prototype – It creates a new instance each time the bean is requested. Spring doesn't manage the full lifecycle, only creating and configuring the bean. Cleanup is the user's responsibility.

Bean Scope

Request – One instance per HTTP request (only in web-aware ApplicationContext)

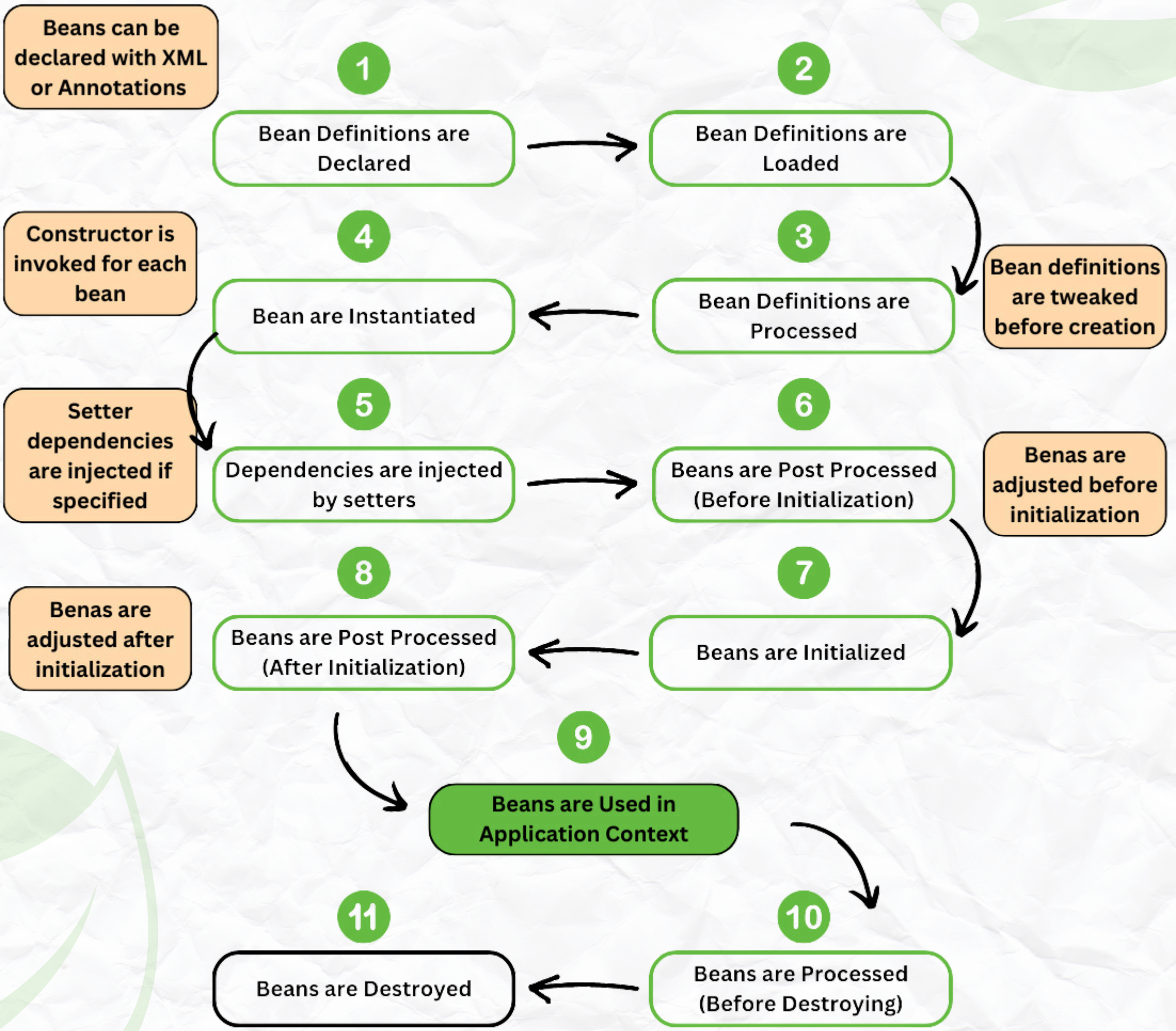
Session – One instance per HTTP session (only in web-aware ApplicationContext)

Application – One instance per ServletContext (only in web-aware ApplicationContext).

Websocket – One instance per WebSocket (only in web-aware ApplicationContext).

LAHIRU
LIYANAPATHIRANA

Bean Life Cycle



LAHIRU
LIYANAPATHIRANA

Bean Definition Declaration

Beans can be defined:

- Via XML
- Via Annotations

Bean Definition Loading

BeanDefinitionReader is responsible for reading and loading the bean definitions.

It has the flexibility to parse and read bean definitions provided by different definition methods.

The loaded beans are then registered with the BeanDefinitionRegistry.

Bean Definition Post Processing

The bean definitions can be further modified before they are instantiated after the beans are loaded.

This can be done by an interface called **BeanFactoryPostProcessor**.

BeanFactoryPostProcessor can:

- modify
- replace
- add new bean definitions

This is useful in altering bean definitions based on environment or external configurations.

Bean Instantiation

BeanFactory (or Application Context) calls the constructor for each bean in the bean registry and instantiates the beans.

BeanFactory creates and injects the required dependencies to the beans.

Bean Setter Injection

After the constructor injection of the dependencies, setter injection takes place.

The XML defined beans are available, they are configured through setter injection.

LAHIRU
LIYANAPATHIRANA

Bean Post Processing (Pre-Initialization)

After the beans are instantiated they are processed before initialization.

The **BeanPostProcessor** interface can be used to modify the beans customly.

This interface has 02 methods:

- postProcessBeforeInitialization
- postProcessAfterInitialization

The **postProcessBeforeInitialization** method is called before initialization.

It returns the bean after modification.

Bean Initialization

Spring allows custom code to be run during the bean initialization phase after the bean properties are set.

They are Post Initialization Callbacks.

They can be defined as follows:

- Methods with the **@PostConstruct** annotation
- Implementing **InitializingBean** interface and calling the method **afterPropertiesSet**()
- With the **init-method** in @Bean annotation definition or XML definition

Bean Initialization

It is possible to define all of the above 03 initialization callbacks for a particular bean.

In this case, Spring invokes the callbacks in the following order:

- @PostConstruct method
- afterPropertiesSet() method
- init-method

Bean Post Processing (Post Initialization)

After initialization, **BeanPostProcessor** interface can be used to further modify the beans.

This second pass by **BeanPostProcessor** occurs, specifically calling the ***postProcessAfterInitialization*** method.

This method can be used to make a proxy for the given bean.

Bean In Use

The bean is in use by the Application Context.

Bean Destruction

During the bean destruction process, Spring allows custom cleanup to be run before the destruction.

They are Pre Destruction Callbacks.

They can be defined as follows:

- Methods with the **@PreDestroy** annotation
- Implementing **DisposableBean** interface and calling the method **destroy**()
- With the **destroy-method** in @Bean annotation definition or XML definition

Bean Destruction

Similar to post initialization callbacks, it is possible to define all of the above 03 destroy callbacks for a particular bean.

In this case, Spring invokes the callbacks in the following order:

- @PreDestroy method
- destroy() method
- destroy-method

Finally, the Spring container removes the bean from its context and performs any final cleanup.

Aware Interfaces

Aware Interfaces in the Spring allow beans to gain access to certain IoC container functionalities and resources.

Following are Aware Interfaces:

ApplicationContextAware
Method: setApplicationContext()
Purpose: Sets the ApplicationContext that the bean runs in.

ApplicationEventPublisherAware
Method: setApplicationEventPublisher()
Purpose: Sets the ApplicationEventPublisher that the bean runs in.

BeanNameAware
Method: setBeanName()
Purpose: Sets the name of the bean in the bean factory.

Aware Interfaces

LoadTimeWeaverAware
Method: setLoadTimeWeaver()
Purpose: Sets the LoadTimeWeaver of this object's containing ApplicationContext.

MessageSourceAware
Method: setMessageSource()
Purpose: Sets the MessageSource that this bean runs in.

NotificationPublisherAware
Method: setNotificationPublisher()
Purpose: Sets the NotificationPublisher instance for the current managed resource instance.

ResourceLoaderAware
Method: setResourceLoader()
Purpose: Sets the ResourceLoader that this object runs in.

ServletContextAware
Method: setServletContext()
Purpose: Sets the ServletContext that this object runs in.

Did You Find This Post Useful?

Stay Tuned for More Spring Related Posts Like This

LAHIRU
LIYANAPATHIRANA