**1. Bean Life Cycle Phases**

The lifecycle consists of **five** main phases:  
1️⃣ **Instantiation** – Object creation.  
2️⃣ **Dependency Injection** – Dependencies are injected.  
3️⃣ **Post-Initialization Processing** – Custom initialization logic.  
4️⃣ **Ready for Use** – The bean is in use.  
5️⃣ **Destruction** – Cleanup before shutting down.

**2. Bean Life Cycle Methods**

Spring provides multiple ways to manage bean lifecycle events:

**2.1 Using @PostConstruct and @PreDestroy (Recommended)**

These annotations from **javax.annotation** provide an easy way to hook into lifecycle events.

java

CopyEdit

import jakarta.annotation.PostConstruct;

import jakarta.annotation.PreDestroy;

import org.springframework.stereotype.Component;

@Component

public class MyBean {

public MyBean() {

System.out.println("1. Bean Instantiated");

}

@PostConstruct

public void init() {

System.out.println("2. @PostConstruct - Bean is initialized");

}

@PreDestroy

public void destroy() {

System.out.println("3. @PreDestroy - Bean is about to be destroyed");

}

}

**Execution Order**

1️⃣ MyBean object is **created**.  
2️⃣ @PostConstruct runs **after dependencies are injected**.  
3️⃣ @PreDestroy runs **before Spring shuts down**.

**2.2 Using InitializingBean and DisposableBean (Legacy)**

Spring provides interfaces for initialization and destruction logic.

java

CopyEdit

import org.springframework.beans.factory.DisposableBean;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.stereotype.Component;

@Component

public class MyBean implements InitializingBean, DisposableBean {

public MyBean() {

System.out.println("1. Bean Instantiated");

}

@Override

public void afterPropertiesSet() throws Exception {

System.out.println("2. afterPropertiesSet() - Bean is initialized");

}

@Override

public void destroy() throws Exception {

System.out.println("3. destroy() - Bean is about to be destroyed");

}

}

**Execution Order**

1️⃣ Constructor is called (**Bean Instantiated**).  
2️⃣ afterPropertiesSet() runs after dependencies are set.  
3️⃣ destroy() is called before bean destruction.

**🚀 Best Practice:** Use @PostConstruct and @PreDestroy instead of these interfaces.

**2.3 Using init-method and destroy-method in XML**

If using **XML-based configuration**, define lifecycle methods in beans.xml.

xml

CopyEdit

<bean id="myBean" class="com.example.MyBean"

init-method="initMethod"

destroy-method="cleanupMethod"/>

And in the Java class:

java

CopyEdit

public class MyBean {

public void initMethod() {

System.out.println("Bean initialized using init-method");

}

public void cleanupMethod() {

System.out.println("Bean destroyed using destroy-method");

}

}

**3. Bean Scope and Lifecycle**

The lifecycle **depends on the bean scope**:

**3.1 Singleton Scope (Default)**

* Only **one instance** per Spring container.
* Lifecycle is managed by the container.

java

CopyEdit

@Component

@Scope("singleton")

public class SingletonBean {

// Lifecycle follows normal singleton rules.

}

**3.2 Prototype Scope**

* A **new instance** is created each time the bean is requested.
* **Destroy method is NOT called automatically**.

java

CopyEdit

@Component

@Scope("prototype")

public class PrototypeBean {

@PostConstruct

public void init() {

System.out.println("Prototype Bean Initialized");

}

@PreDestroy

public void destroy() {

System.out.println("Prototype Bean Destroyed"); // This won't be called automatically!

}

}

**⚠ Important:**  
For prototype beans, you must manually handle destruction:

java

CopyEdit

ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

PrototypeBean bean = context.getBean(PrototypeBean.class);

((ConfigurableApplicationContext) context).close(); // Destroy NOT called for prototype beans.

**4. Complete Execution Order of Bean Life Cycle**

1️⃣ **Spring Container Instantiates the Bean** (via new).  
2️⃣ **Dependencies are Injected** (@Autowired fields are set).  
3️⃣ **@PostConstruct or afterPropertiesSet() is called**.  
4️⃣ **Bean is ready for use**.  
5️⃣ **On shutdown, @PreDestroy or destroy() is called**.

**5. Spring Bean Life Cycle with BeanPostProcessor**

Spring allows **modifying a bean before and after initialization** using BeanPostProcessor.

java

CopyEdit

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.config.BeanPostProcessor;

import org.springframework.stereotype.Component;

@Component

public class CustomBeanProcessor implements BeanPostProcessor {

@Override

public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {

System.out.println("Before Initialization: " + beanName);

return bean;

}

@Override

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {

System.out.println("After Initialization: " + beanName);

return bean;

}

}

**Execution Order with BeanPostProcessor**

1️⃣ Constructor is called → Bean Instantiated  
2️⃣ postProcessBeforeInitialization() executes  
3️⃣ @PostConstruct runs  
4️⃣ postProcessAfterInitialization() executes  
5️⃣ **Bean is ready for use**  
6️⃣ On shutdown, @PreDestroy runs  
7️⃣ postProcessBeforeDestruction() executes

**6. Summary Table**

| **Approach** | **Method Called** | **When?** |
| --- | --- | --- |
| **@PostConstruct** | init() | After dependencies are injected |
| **@PreDestroy** | destroy() | Just before bean destruction |
| **InitializingBean** | afterPropertiesSet() | After dependencies are set |
| **DisposableBean** | destroy() | Before bean is removed |
| **XML init-method** | Custom method | After bean initialization |
| **XML destroy-method** | Custom method | Before bean destruction |
| **BeanPostProcessor** | postProcessBeforeInitialization() | Before @PostConstruct |
| **BeanPostProcessor** | postProcessAfterInitialization() | After @PostConstruct |

**7. Best Practices**

✅ Use @PostConstruct and @PreDestroy instead of InitializingBean and DisposableBean.  
✅ For prototype beans, handle destruction manually.  
✅ Use BeanPostProcessor for modifying beans globally.  
✅ Avoid overriding lifecycle methods unless necessary.