Bean Life Cycle in Java Spring

Last Updated : 04 Aug, 2025

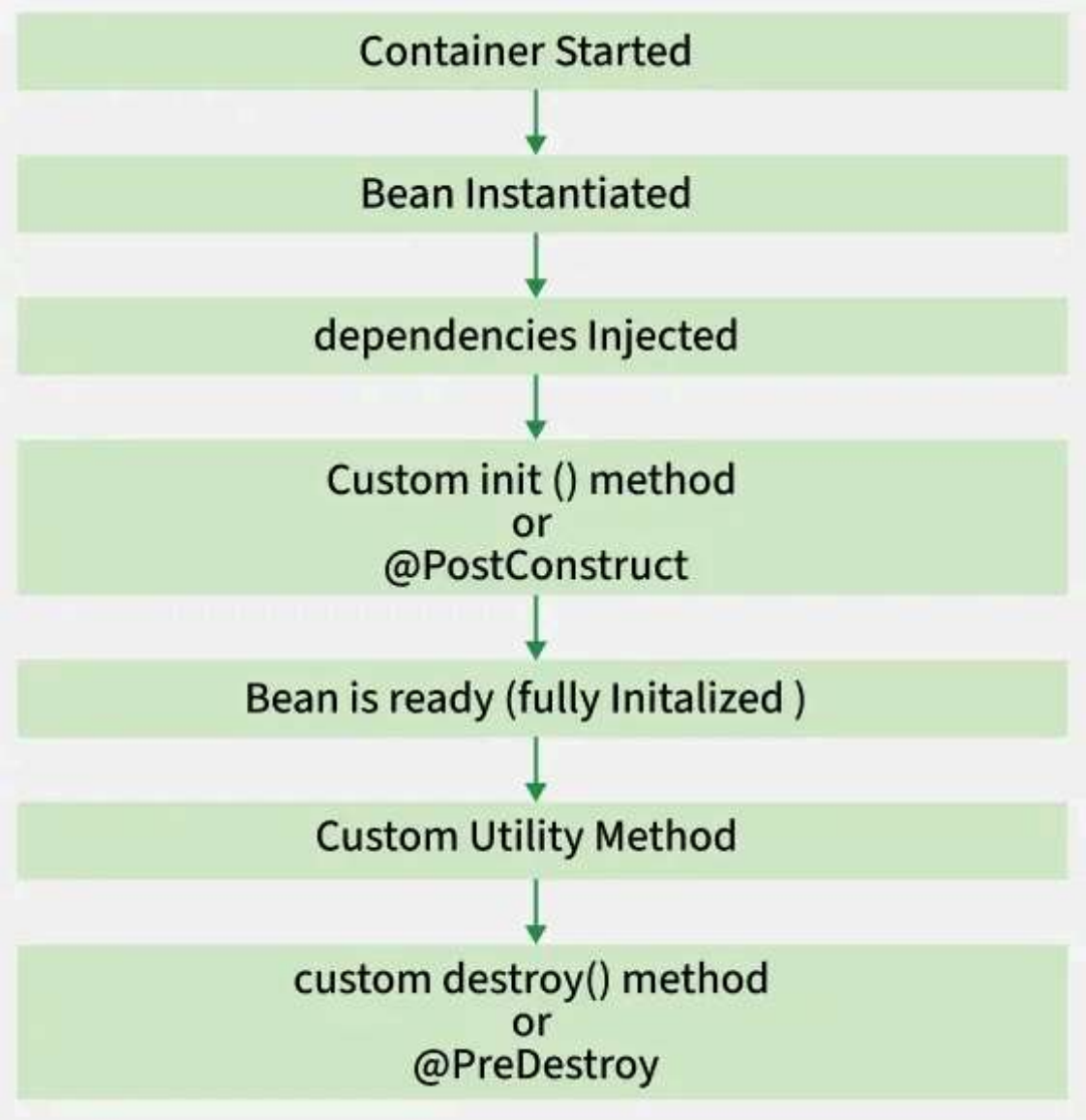
The bean lifecycle in Spring is the sequence of steps a bean goes through from creation to destruction and it's managed by the Spring container.

Bean Life Cycle Phases

The lifecycle of a Spring bean consists of the following phases, which are listed below

- **Container Started:** The Spring IoC container is initialized.
- **Bean Instantiated:** The container creates an instance of the bean.
- **Dependencies Injected:** The container injects the dependencies into the bean.
- **Custom init() method:** If the bean implements InitializingBean or has a custom initialization method specified via @PostConstruct or init-method.
- **Bean is Ready:** The bean is now fully initialized and ready to be used.
- **Custom utility method:** This could be any custom method you have defined in your bean.
- **Custom destroy() method:** If the bean implements DisposableBean or has a custom destruction method specified via @PreDestroy or destroy-method, it is called when the container is shutting down.

The below image demonstrates the Spring bean lifecycle:



Note: We can choose a custom method name instead of init() and destroy(). Here, we will use init() method to execute all its code as the spring container starts up and the bean is instantiated and destroy() method to execute all its code on closing the container.

Ways to Implement the Bean Life Cycle

Spring provides three ways to implement the life cycle of a bean. In order to understand these three ways, let's take an example. In this example, we will write and activate init() and destroy() method for our bean (HelloWorld.java) to print some messages on start and close of the Spring container. Therefore, the three ways to implement this are:

1. Using XML Configuration

In this approach, in order to avail custom init() and destroy() methods for a bean we have to register these two methods inside the Spring XML configuration file while defining a bean. Therefore, the following steps are followed:

Step1: Create the bean Class

Firstly, we need to create a bean **HelloWorld.java** in this case and write the init() and destroy() methods in the class.

```
package beans;

public class HelloWorld {

    // Custom init method
    public void init() throws Exception {
        System.out.println("Bean HelloWorld has been instantiated, and I'm
the init() method");
    }

    // Custom destroy method
    public void destroy() throws Exception {
        System.out.println("Container has been closed, and I'm the destroy()
method");
    }
}
```

Step 2: Configure the Spring XML File

Now, we need to configure the spring XML file **spring.xml** and need to register the init() and destroy() methods in it.

```
<beans xmlns="http://www.springframework.org/schema/beans//////"
       xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans//////
           http://www.springframework.org/schema/beans///////spring-
beans.xsd">

    <bean id="hw" class="beans.HelloWorld"
          init-method="init" destroy-method="destroy"/>
</beans>
```

Step 3: Create a Driver Class

We need to create a driver class to run this bean.

```
package test;
import beans.HelloWorld;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {
    public static void main(String[] args) throws Exception {
        // Loading the Spring XML configuration file into the spring
container and it will create the instance of the bean as it loads into
container
        ConfigurableApplicationContext cap = new
ClassPathXmlApplicationContext("resources/spring.xml");

        cap.close();
    }
}
```

Output:

```
Bean HelloWorld has been instantiated and I'm the init() method
Container has been closed and I'm the destroy() method
```

2. Using Programmatic Approach (Interface)

To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need to implement our bean with two interfaces.

In this approach, we implement the InitializingBean and DisposableBean interfaces and override their methods afterPropertiesSet() and destroy() method is invoked just after the container is closed.

> ***Note:*** *To invoke destroy method we have to call a* ***close()*** *method of ConfigurableApplicationContext.*

Step 1: Create the Bean Class

We need to create a bean class HelloWorld, by implementing InitializingBean, DisposableBean and overriding afterPropertiesSet() and destroy() method.

```
package beans;

import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class HelloWorld implements InitializingBean,DisposableBean {

    @Override
    public void afterPropertiesSet() throws Exception
    {
        System.out.println("Bean HelloWorld has been "  + "instantiated and
I'm the " + "init() method");
    }

    @Override
    public void destroy() throws Exception
    {
        System.out.println("Container has been closed "+ "and I'm the
destroy() method");
    }
}
```

Step 2: Configure the Spring XML File

Now, we need to configure the spring XML file **spring.xml** and define the bean.

```
<beans xmlns="http://www.springframework.org/schema/beans//////"
       xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans//////
           http://www.springframework.org/schema/beans///////spring-
beans.xsd">

    <bean id="hw" class="beans.HelloWorld"/>
</beans>
```

Step 3: Create a Driver Class

Finally, we need to create a driver class to run this bean.

```
package test;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import beans.HelloWorld;

public class Client {

    public static void main(String[] args) throws Exception
    {

        ConfigurableApplicationContext cap = new
```

```
ClassPathXmlApplicationContext("resources/spring.xml");
        cap.close();
    }
}
```

Output:

```
Bean HelloWorld has been instantiated and I'm the init() method
Container has been closed and I'm the destroy() method
```

3. Using Annotations

To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need to annotate init() method by @PostConstruct annotation and destroy() method by @PreDestroy annotation.

> ***Note:*** *To invoke the destroy() method we have to call the close() method of ConfigurableApplicationContext.*

Step 1: Create the Bean Class

We need to create a bean HelloWorld.java in this case and annotate the custom init() method with @PostConstruct and destroy() method with @PreDestroy.

```
package beans;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class HelloWorld {

    @PostConstruct
    public void init() throws Exception
    {
        System.out.println(
            "Bean HelloWorld has been " + "instantiated and I'm the " +
"init() method");
    }

    @PreDestroy
```

```
    public void destroy() throws Exception
    {
        System.out.println("Container has been closed " + "and I'm the
destroy() method");
    }
}
```

Step 2: Configure the Spring XML File

Now, we need to configure the spring XML file spring.xml and define the bean.

```
<beans xmlns="http://www.springframework.org/schema/beans//////"
       xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans//////
           http://www.springframework.org/schema/beans///////spring-
beans.xsd">

    <!-- Activate the @PostConstruct and @PreDestroy annotations -->
    <bean
class="org.springframework.context.annotation.CommonAnnotationBeanPostProces
sor"/>

    <!-- Configure the bean -->
    <bean id="hw" class="beans.HelloWorld"/>

</beans>
```

Step 3: Create a Driver Class

Finally, we need to create a driver class to run this bean.

```
package test;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import beans.HelloWorld;

public class Client {

    public static void main(String[] args) throws Exception
    {
        ConfigurableApplicationContext cap
            = new ClassPathXmlApplicationContext(
                "resources/spring.xml");

        cap.close();
    }
}
```

Output: