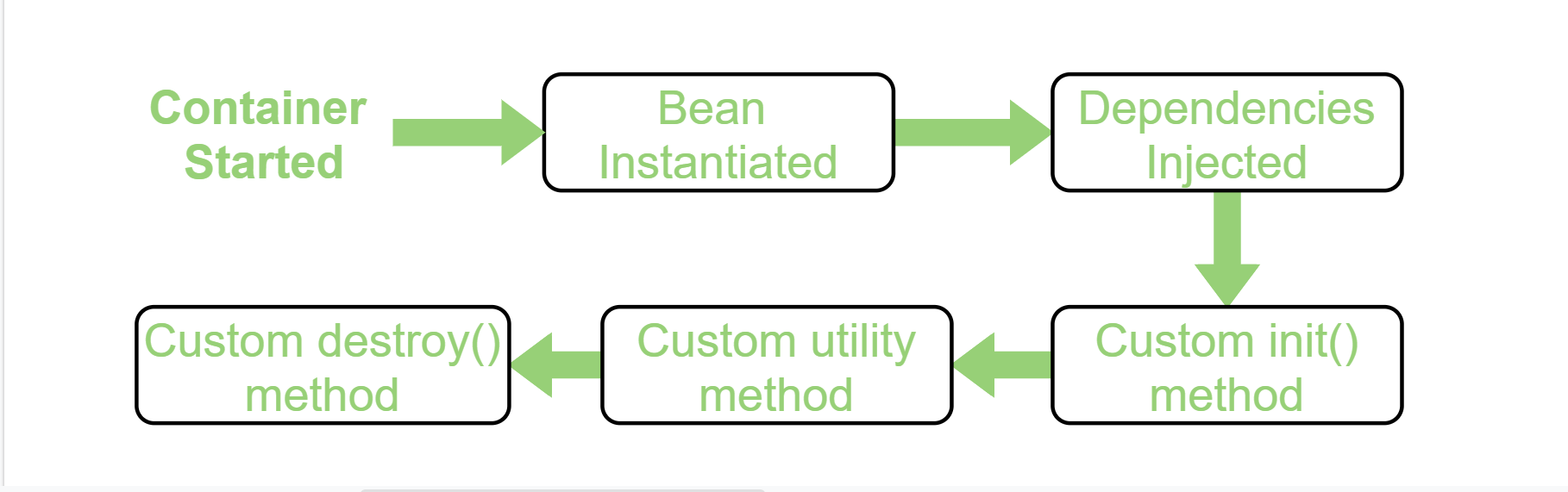
### Bean Life Cycle

The lifecycle of any object means when & how it is born, how it behaves throughout its life, and when & how it dies. Similarly, the bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed. In this article, we will discuss the life cycle of the bean.

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom **init()** method and the **destroy()** method.

The following image shows the process flow of the bean life cycle.

[](https://media.geeksforgeeks.org/wp-content/uploads/20200428011831/Bean-Life-Cycle-Process-flow3.png)

*Bean Life Cycle Process Flow*

**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 life cycle of a bean****

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. By XML:** 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:

**Step 1:** Firstly, we need to create a bean **HelloWorld.java** in this case and write the init() and destroy() methods in the class.

* Java

|  |
| --- |
| // Java program to create a bean  // in the spring framework  **package** beans;    **public** **class** HelloWorld {        // This method executes      // automatically as the bean      // is instantiated  **public** **void** init() **throws** Exception      {          System.out.println(              "Bean HelloWorld has been "              + "instantiated and I'm "              + "the init() method");      }        // This method executes      // when the spring container      // is closed  **public** **void** destroy() **throws** Exception      {          System.out.println(              "Container has been closed "              + "and I'm the destroy() method");      }  } |

**Step 2:** Now, we need to configure the spring XML file **spring.xml** and need to register the init() and destroy() methods in it.

* XML

|  |
| --- |
| <!DOCTYPE      beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"          "<http://www.springframework.org/dtd/spring-beans-2.0.dtd>">    <**beans**>      <**bean** id="hw" class="beans.HelloWorld"              init-method="init" destroy-method="destroy"/>    </**beans**> |

**Step 3:** Finally, we need to create a driver class to run this bean.

* Java

|  |
| --- |
| // Java program to call the  // bean initialized above    **package** test;    **import** beans.HelloWorld;  **import** org.springframework.context.ConfigurableApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;    // Driver class  **public** **class** Client {        // Main driver method  **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");            // It will close the spring container          // and as a result invokes the          // destroy() method          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. By Programmatic Approach:** 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 namely **InitializingBean**, **DisposableBean** and will have to override **afterPropertiesSet()** and **destroy()** method. **afterPropertiesSet()** method is invoked as the container starts and the bean is instantiated whereas, the **destroy()** method is invoked just after the container is closed.

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

Therefore, the following steps are followed:

* Firstly, we need to create a bean **HelloWorld.java** in this case by implementing InitializingBean, DisposableBean, and overriding afterPropertiesSet() and destroy() method.
* Java

|  |
| --- |
| // Java program to create a bean  // in the spring framework  **package** beans;    **import** org.springframework      .beans.factory.DisposableBean;    **import** org.springframework      .beans.factory.InitializingBean;    // HelloWorld class which implements the  // interfaces  **public** **class** HelloWorld  **implements** InitializingBean,   DisposableBean {        @Override      // It is the init() method      // of our bean and it gets      // invoked on bean instantiation  **public** **void** afterPropertiesSet()  **throws** Exception      {          System.out.println(              "Bean HelloWorld has been "              + "instantiated and I'm the "              + "init() method");      }        @Override      // This method is invoked      // just after the container      // is closed  **public** **void** destroy() **throws** Exception      {          System.out.println(              "Container has been closed "              + "and I'm the destroy() method");      }  } |

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

|  |
| --- |
| <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"              "<http://www.springframework.org/dtd/spring-beans-2.0.dtd>">    <**beans**>      <**bean** id="hw" class="beans.HelloWorld"/>    </**beans**> |

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

|  |
| --- |
| // Java program to call the  // bean initialized above    **package** test;    **import** org.springframework      .context      .ConfigurableApplicationContext;    **import** org.springframework      .context.support      .ClassPathXmlApplicationContext;    **import** beans.HelloWorld;    // Driver class  **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");            // It will close the spring container          // and as a result invokes the          // destroy() method          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 Annotation:**

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.*

Therefore, the following steps are followed:

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

|  |
| --- |
| // Java program to create a bean  // in the spring framework  **package** beans;    **import** javax.annotation.PostConstruct;  **import** javax.annotation.PreDestroy;    // HelloWorld class  **public** **class** HelloWorld {        // Annotate this method to execute it      // automatically as the bean is      // instantiated      @PostConstruct  **public** **void** init() **throws** Exception      {          System.out.println(              "Bean HelloWorld has been "              + "instantiated and I'm the "              + "init() method");      }        // Annotate this method to execute it      // when Spring container is closed      @PreDestroy  **public** **void** destroy() **throws** Exception      {          System.out.println(              "Container has been closed "              + "and I'm the destroy() method");      }  } |

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

|  |
| --- |
| <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"              "<http://www.springframework.org/dtd/spring-beans-2.0.dtd>">    <**beans**>        <!-- activate the @PostConstruct and  @PreDestroy annotation -->        <**bean** class="org.springframework  .context.annotation  .CommonAnnotationBeanPostProcessor"/>        <!-- configure the bean -->      <**bean** class="beans.HelloWorld"/>    </**beans**> |

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

|  |
| --- |
| // Java program to call the  // bean initialized above    **package** test;    **import** org.springframework.context.ConfigurableApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** beans.HelloWorld;    // Driver class  **public** **class** Client {        // Main driver method  **public** **static** **void** main(String[] args) **throws** Exception      {            // Loading the Spring XML configuration          // file into Spring container and          // it will create the instance of the          // bean as it loads into container          ConfigurableApplicationContext cap              = **new** ClassPathXmlApplicationContext(                  "resources/spring.xml");            // It will close the Spring container          // and as a result invokes the          // destroy() method          cap.close();      }  } |

**Output:**

Bean HelloWorld has been instantiated and I'm the init() method

Container has been closed and I'm the destroy() method

## How to Create a Spring Bean in 3 Different Ways?

Spring is one of the most popular Java EE frameworks. It is an open-source lightweight framework that allows Java EE 7 developers to build simple, reliable, and scalable enterprise applications. This framework mainly focuses on providing various ways to help you manage your business objects. It made the development of Web applications much easier than compared to classic Java frameworks and application programming interfaces (APIs), such as Java database connectivity (JDBC), JavaServer Pages(JSP), and Java Servlet. 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 otherwise managed by a Spring IoC container.

**Different Methods to Create a Spring Bean**

Here we are going to discuss how to create a Spring Bean in 3 different ways as follows:

* Creating Bean Inside an XML Configuration File (beans.xml)
* Using @Component Annotation
* Using @Bean Annotation

**Method 1: Creating Bean Inside an XML Configuration File (beans.xml)**

One of the most popular ways to create a spring bean is to define a bean in an XML configuration file something like this.

<bean id="AnyUniqueId" class="YourClassName">

</bean>

Let us create a simple class Student having two attributes id and studentName and later creating a simple method to print the details of the student.

**Example**

|  |
| --- |
| // Java Program to Illustrate Student Class    // Class  public class Student {        // Class data members      private int id;      private String studentName;        // Method      public void displayInfo()      {          // Print statement          System.out.println("Student Name is " + studentName                             + " and Roll Number is " + id);      }  } |

Now let’s create an XML file named beans.xml file in the project classpath. And inside this beans.xml file, we have to define our Student bean something like this. And that’s it. In this way, you can create beans in spring.

Example

XML

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="<http://www.springframework.org/schema/beans>"         xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"         xsi:schemaLocation="<http://www.springframework.org/schema/beans>  <https://www.springframework.org/schema/beans/spring-beans.xsd>">        <bean id="studentAmiya" class="Student">        </bean>    </beans> |

**Method 2: Using @Component Annotation**

Spring Annotations are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program. @Component is an annotation that allows Spring to automatically detect the custom beans.

Example: Suppose we have already a Java project and all the Spring JAR files are imported into that project. Now let’s create a simple class named College and inside the class, we have a simple method. Below is the code for the College.java file.

A. File: College.java

Java

|  |
| --- |
| // Java Program to Illustrate College Class    package ComponentAnnotation;    // Class  public class College {        // Method      public void test()      {          // Print statement          // whenever this method is called          System.out.println("Test College Method");      }  } |

Now let’s create a Bean for this class. So we can use @Component annotation for doing the same task. So we can modify our College.java file something like this. And that’s it.

B. College.java

Java

|  |
| --- |
| // Java Program to Illustrate College Class    package ComponentAnnotation;    // Importing required classes  import org.springframework.stereotype.Component;    @Component("collegeBean")    // Class  public class College {        // Method      public void test()      {          // Print statement          System.out.println("Test College Method");      }  } |

**Method 3: Using @Bean Annotation**

One of the most important annotations in spring is the @Bean annotation which is applied on a method to specify that it returns a bean to be managed by Spring context. Spring Bean annotation is usually declared in Configuration classes methods.

Suppose we have already a Java project and all the Spring JAR files are imported into that project. Now let’s create a simple class named College and inside the class, we have a simple method. Below is the code for the College.java file.

**College.java**

|  |
| --- |
| package BeanAnnotation;    import org.springframework.stereotype.Component;    public class College {        public void test(){          System.out.println("Test College Method");      }  } |

Now let’s create a Configuration class named CollegeConfig.

 CollegeConfig.java

|  |
| --- |
| **package ComponentAnnotation;**    **import org.springframework.context.annotation.ComponentScan;**  **import org.springframework.context.annotation.Configuration;**    **@Configuration**  **public class CollegeConfig {**    **}** |

Here, we are going to create the spring beans using the @Bean annotation. To create the College class bean using the @Bean annotation inside the configuration class we can write something like this inside our CollegeConfig.java file. Please refer to the comments for a better understanding.

**@Bean**

**// Here the method name is the**

**// bean id/bean name**

**public College collegeBean(){**

**// Return the College object**

**return new College();**

**}**

Implementation: Below is the complete code for the CollegeConfig.java file that is below as follows:

Java

|  |
| --- |
| // Java Program to Illustrate Configuration in College Class    package BeanAnnotation;    // Importing required classes  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;    @Configuration  public class CollegeConfig {        // Using Bean annotation to create      // College class Bean      @Bean        // Here the method name is the      // bean id/bean name      public College collegeBean()      {            // Return the College object          return new College();      }  } |