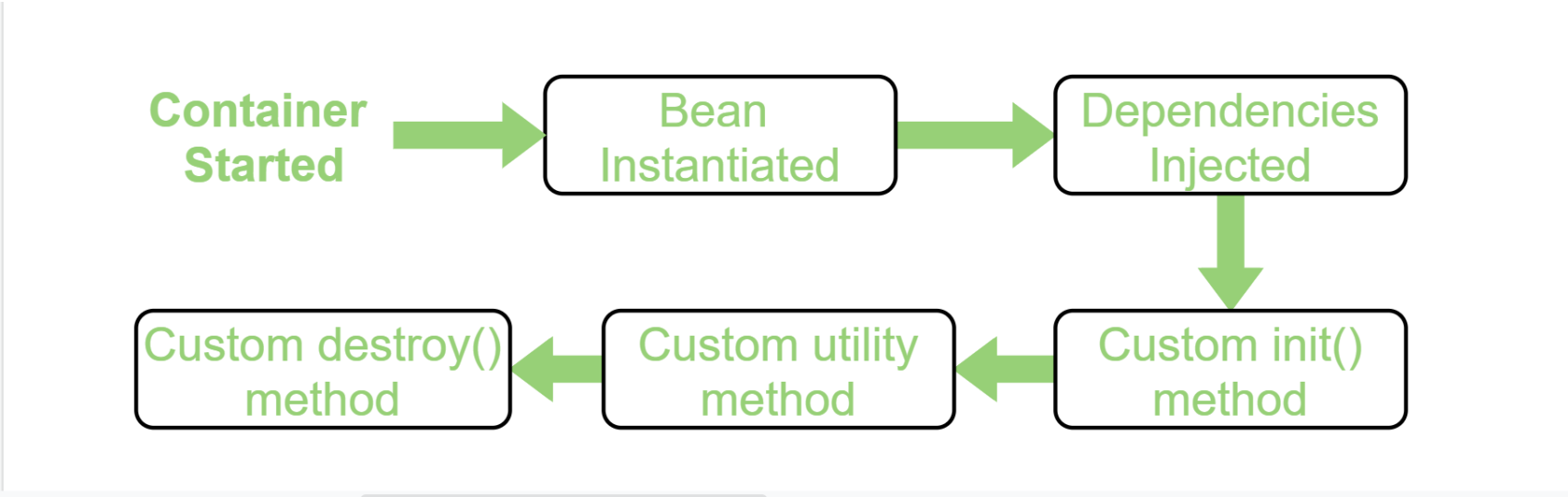
Bean life cycle in Java Spring

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.



Bean Life Cycle Process Flow

Note: We can choose 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 message on start and close of Spring container. Therefore, the three ways to implement this are:

1. **By XML:** In this approach, in order to avail custom init() and destroy() method for a bean we have to register these two methods inside Spring XML configuration file while defining a bean. Therefore,

the following steps are followed:

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

```
// 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(
            "Conatiner has been closed "
            + "and I'm the destroy() method");
    }
}
```

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

```
<!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>
```

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

```
// 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*
> *Conatiner 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 container is closed.
Note: To invoke destroy method we have to call **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 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(
                "Conatiner has been closed "
                + "and I'm the destroy() method");
    }
}
```

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

```
<!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 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
Conatiner 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 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 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(
            "Conatiner has been closed "
            + "and I'm the destroy() method");
    }
}
```

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

```
<!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"/>
```

```
    <!-- cofigure the bean -->
    <bean class="beans.HelloWorld"/>

</beans>
```

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

```
// 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 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
Conatiner has been closed and I'm the destroy() method