**9. Spring Configuration with Java Annotations - Lifecycle Methods**

**Bean Lifecycle Methods / Hooks**:

1. We can add custom code during bean initialization
   1. Calling custom business logic methods
   2. Setting up handles to resources (db, sockets, file etc.)
2. We can add custom code during bean destruction
   1. Calling custom business logic method
   2. Clean up handles to resources (db, sockets, files etc.)

**Development Process**:

1. Define method for init and destroy
2. Add annotations: @PostConstruct and @PreDestroy

**Init: method configuration**:

Code will execute after constructor and after injection of dependencies

@Component

public class TennisCoach implements Coach {

@PostConstruct

public void doMyStartupStuff() { … }

...

}

**Destroy: method configuration**:

Code will execute before bean destroyed.

@Component

public class TennisCoach implements Coach {

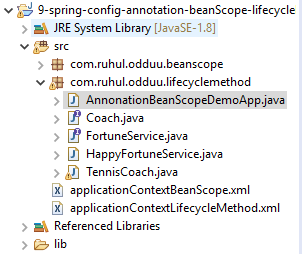
@PreDestroy

public void doMyCleanupStuff() { … }

...

}

**Example**:



**File: Coach.java**:

**package** com.ruhul.odduu.lifecyclemethod;

**public** **interface** Coach {

**public** String getDailyWorout();

**public** String getDailyFortune();

}

**File: FortuneService.java**:

**package** com.ruhul.odduu.lifecyclemethod;

**public** **interface** FortuneService {

**public** String getFortune();

}

**File: HappyFortuneService.java**:

**package** com.ruhul.odduu.lifecyclemethod;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** HappyFortuneService **implements** FortuneService {

@Override

**public** String getFortune() {

**return** "Today is your lucky day!!!";

}

}

**File: TennisCoach.java**:

**package** com.ruhul.odduu.lifecyclemethod;

**import** javax.annotation.PostConstruct;

**import** javax.annotation.PreDestroy;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** TennisCoach **implements** Coach {

// field injection

@Autowired

**private** FortuneService fortuneService;

// default constructor

**public** TennisCoach() {

System.***out***.println(">> TennisCoach: inside default constructor");

}

// code will execute after constructor and after injection of dependencies

// define my init method

@PostConstruct

**public** **void** doMyStartupStuff() {

System.***out***.println(">> TennisCoach: inside of constructor doMyStartupStuff()");

}

// code will execute before bean destroyed

// define my destroy method

@PreDestroy

**public** **void** doMyCleanupStuff() {

System.***out***.println(">> TennisCoach: inside of constructor doMyCleanupStuff()");

}

@Override

**public** String getDailyWorout() {

**return** "Practice your backhand volley";

}

@Override

**public** String getDailyFortune() {

**return** fortuneService.getFortune();

}

}

**File: applicationContextLifecycleMethod.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"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- add entry to enable component scanning -->

<context:component-scan

base-package=*"com.ruhul.odduu.lifecyclemethod"* />

</beans>

**File: AnnonationBeanScopeDemoApp.java**:

**package** com.ruhul.odduu.lifecyclemethod;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** AnnonationBeanScopeDemoApp {

**public** **static** **void** main(String[] args) {

// load spring configuration file

ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext(

"applicationContextLifecycleMethod.xml");

// get the bean from spring container

Coach theCoach = context.getBean("tennisCoach", Coach.**class**);

// call a method on the bean

System.***out***.println(theCoach.getDailyWorout());

// call method to get the daily fortune

System.***out***.println(theCoach.getDailyFortune());

// close the context

context.close();

}

}

/\*

**Output**:

>> TennisCoach: inside default constructor

>> TennisCoach: inside of constructor doMyStartupStuff() // @PostConstruct

Practice your backhand volley

Today is your lucky day!!!

>> TennisCoach: inside of constructor doMyCleanupStuff() // @PreDestroy

\*/

**Special Note about @PostConstruct and @PreDestroy Method Signatures**:

Here is some additional details information about the method signatures of @PostContruct and @PreDestroy methods.

1. **Access Modifiers**:

The method can have any access modifier (public, protected, private)

1. **Return type**:

The method can have any return type. However, "void' is most commonly used. If you give a return type just note that you will not be able to capture the return value. As a result, "void" is commonly used.

1. **Method Name**:

The method can have any method name.

1. **Arguments**:

The method cannot accept any arguments. The method should be no-arg.

**HEADS UP - FOR JAVA 9, 10 and 11 USERS - @PostConstruct and @PreDestroy**:

If we are using Java 9, 10 or 11, then we will encounter an error when using @PostConstruct and @PreDestroy in our code.

**Error**:

Eclipse is unable to import @PostConstruct or @PreDestroy

This happens because of Java 9 and higher.

When using Java 9 and higher, javax.annotation has been removed from its default classpath. That's why Eclipse can't find it.

For solve this problem we have to follow the following steps.

**Solution**:

1. Download the javax.annotation-api-1.2.jar from

<http://central.maven.org/maven2/javax/annotation/javax.annotation-api/1.2/javax.annotation-api-1.2.jar>

1. Copy the JAR file to the lib folder of our project.
2. Right-click your project, select

Properties => Java Build Path => Libraries => Classpath => Add JARs ...

1. Navigate to the JAR file <our-project>/lib/javax.annotation-api-1.2.jar
2. Click OK then click Apply and Close

Eclipse will perform a rebuild of your project and it will resolve the related build errors.

**Special Note about Destroy Lifecycle and Prototype Scope**:

For "prototype" scoped beans, Spring does not call the @PreDestroy method.

In contrast to the other scopes, Spring does not manage the complete lifecycle of a prototype bean: the container instantiates, configures, and otherwise assembles a prototype object, and hands it to the client, with no further record of that prototype instance.

Thus, although initialization lifecycle callback methods are called on all objects regardless of scope, in the case of prototypes, configured destruction lifecycle callbacks are not called. The client code must clean up prototype-scoped objects and release expensive resources that the prototype bean(s) are holding.

To get the Spring container to release resources held by prototype-scoped beans, try using a custom bean post-processor, which holds a reference to beans that need to be cleaned up.

---

This also applies to XML configuration.

9. Spring Configuration with Java Annotations - Lifecycle Methods