# Spring Bean Life Cycle

Keeping track of various activity from birth to death is called life cycle.

Servlet container will take care of servlet component life cycle by raising 3 life cycle events and calling 3 life cycle methods for those life cycle events.

Events are:

|  |  |
| --- | --- |
| ***Life Cycle Event*** | ***Life Cycle Method*** |
| **Instantiation Event** (Raises when servlet container creates servlet component class object) | public void init(ServletConfig cg) |
| **Request Processing Event** (raises when servlet container keeps servlet component class object ready (completing initiation and initialization) to process request) | public void service(ServletRequest req, ServletResponse res) throws ServletException, IO Exception |
| **Destruction Event** (Raises when servlet container is about to destroyed our servlet component/ class/ object | public void destroy() |

Servlet Component is invasive I.e., it must implement *javax.servlet.Servlet(I)***.** So servlet component is having fixed life cycle methods names.

*Spring container/ICO container manages spring bean life cycle i.e, from birth to death (object creation to object distruction) of spring bean class will be taken care by IOC container.*

IOC container raises two life cycle events in spring beans life cycle, they are:

|  |  |
| --- | --- |
| ***Life Cycle Event*** | ***Life Cycle Method*** |
| **Instantiation Event** (Raises when IOC container creates spring bean class object and also completes injections on that object.) | init() method no fixed name |
| **Destruction Event** (Raises when IOC container is about to destroye spring bean class object. | destroy() method no fixed name |

***NOTE: There is no request processing event in spring life cycle because spring bean is not a web component to process the http request***

In the init() life cycle method of spring bean we generally keep:

* Logic to initialize spring bean properties that are not participating in dependency injection.
* Logic to check weather important spring bean properties are injected with correct values of not, if not we will throw exception so that business logic in business method will not be executed with wrong values.

In the destroy() life cycle method of spring bean we generally keep:

* Logic to nullify bean properties.
* Logic to release non-java resources that are associated with spring bean (like closing JDBC connection object and etc..)

## Three approaches of Implementing spring bean lifecycle

1. ***Declarative Approach:*** Configure spring bean life cycle using init-method, destroyed-method attribute of <bean> tag. In this approach spring bean life cycle methods are custome names
2. ***Programmatic Approach:*** Make spring bean class implementing two spring api supplied interfaces:
   1. **Initializing beans**, it has one method which is : **afterPropertySet()**
   2. **Disposable Bean Interface,** it has one method which is : **destroy()**
3. **Annotation Approach**: Configure init-method using **@PostConstruct** and configure destroy-method using **@PreDestroy** annotations. Here spring beans life cycle methods are user defined.

***Declarative Approach:***

* Here spring bean can be invasive or non-invasive
* We must take spring bean life cycle method having the following signature:

|  |
| --- |
| public void <method>(){  } |

* In one spring bean configuration we can have only one init-method and one destroyed life cycle method.
* *NOTE: We must call* ***ctx.close()*** *method explicitly to make IOC container to destroyed single-ton scope spring bean class object.*
* *NOTE: Destroy life cycle method execute on the destruction of spring bean class object only for single ton scope spring beans not for other scopes of spring beans because only singleton scope spring class object will be maintained in the IOC container internal cache.*
* *NOTE: While working with bean factory container, since there is no provision to close bean factory container so we need to call* ***factory.destroySingletone()*** *method to make IOC container to destroy singleton scope spring bean class objects.*
* *NOTE: life cycle methods will not be executed on the order of the availability they will be executed based on the events raised by the container.*

|  |
| --- |
| *Example:* |
| *JAVA BEAN CLASS::*  *package com.ab.springlifecycle.beans;*    *import java.util.Date;*    *public class VoterVerifier {*  *private String name;*  *private float age;*  *private Date verificationDate;*  *public VoterVerifier() {*  *System.out.println("0 :: param constructor VoterVerifier.VoterVerifier()");*  *}*  *// Setter methdos for setter injection*  *public void setName(String name) {*  *System.out.println("VoterVerifier.setName()");*  *this.name = name;*  *}*  *public void setAge(float age) {*  *System.out.println("VoterVerifier.setAge()");*  *this.age = age;*  *}*  *// business method:*  *public String checkVotingEligibility() {*  *if(age<18)*  *return "MR/Miss/Mrs. " + name +" you are not elibile for voting wait " + (18-age) +" to cast your vote";*  *else*  *return "MR/Miss/Mrs. " + name +" you are elibile for voting don't wait varification date: " + verificationDate ;*  *}*  *//-- init-life cycle method*  *public void myInit() {*  *System.out.println("VoterVerifier.myInit()(Custome init Method)");*  *// initilization the value which are not taking particapting in dependency injection*  *verificationDate = new Date();*  *// validation*  *if (name == null || age <= 0) {*  *throw new IllegalArgumentException("Invalid User Input");*  *}*  *}*  */\*for init life cycle method signature should be public void only\*/*  *//- destroyed life cycle method*  *public void myDestroy() {*  *System.out.println("VoterVerifier.myDestroy()");*  *name = null;*  *age = 0;*  *verificationDate = null;*  *}*  *}* |
| *Configuration file (Xml file):*  *<?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 http://www.springframework.org/schema/beans/spring-beans.xsd">*    *<!-- spring bean configuration -->*  *<bean id="voter" class="com.ab.springlifecycle.beans.VoterVerifier" init-method="myInit" destroy-method="myDestroy">*  *<property name="name" value="raja"></property>*  *<property name="age" value="19"></property>*  *</bean>*    *</beans>* |
| *Main class:*  *package com.ab.springlifecycle;*    *import org.springframework.context.support.ClassPathXmlApplicationContext;*    *import com.ab.springlifecycle.beans.VoterVerifier;*    *public class App*  *{*  *public static void main( String[] args ){*  *// create IOC container:*  *ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("com/ab/springlifecycle/cfg/applicationContext.xml");*    *// get spring bean object:*  *VoterVerifier voter = ctx.getBean("voter", VoterVerifier.class);*    *// invokes business method:*  *try {*  *System.out.println(voter.checkVotingEligibility());*  *} catch (Exception e) {*  *e.printStackTrace();*  *}*    *// close the container*  *ctx.close();*  *}*  *}* |

Pros of Declarative approach:

* Here spring can be invasive or non-invasive.
* In user-defined class we can take custome methods as life cycle methods

Cons of Declarative approach:

* We must remember and configuration spring bean life cycle methods in spring beans otherwise they won be executed.
* While configuring pre-define class as spring bean identifying spring bean life cycle method are present or not and configure them by knowing their name is very difficult.

***Programatic Approach:***

* Here spring bean class should implement the spring api supplied interfaces:

|  |
| --- |
| ***Org.springframework.beans.factory. InitializingBean***  public void afterPropertiesSet() throws Exception; |
| ***Org.springframework.beans.factory. DisposableBean***  public void destroy() throws Exception; |

* This approach makes spring bean classes as insavie classes (Tightly coupled classes with spring API.)
* Here no need of configuration life cycle method in spring bean configuration file but they will be called by IOC container automatically based on the interface that spring bean class is implementing.
* Example:

|  |
| --- |
| package com.ab.springlifecycle.beans;    import java.util.Date;    import org.springframework.beans.factory.DisposableBean;  import org.springframework.beans.factory.InitializingBean;    public class VoterVerifier implements InitializingBean, DisposableBean {    private String name;  private float age;  private Date verificationDate;    public VoterVerifier() {  System.out.println("0 :: param constructor VoterVerifier.VoterVerifier()");  }    // Setter methdos for setter injection  public void setName(String name) {  System.out.println("VoterVerifier.setName()");  this.name = name;  }    public void setAge(float age) {  System.out.println("VoterVerifier.setAge()");  this.age = age;  }    // business method:  public String checkVotingEligibility() {  if (age < 18)  return "MR/Miss/Mrs. " + name + " you are not elibile for voting wait " + (18 - age) + " to cast your vote";  else  return "MR/Miss/Mrs. " + name + " you are elibile for voting don't wait varification date: "  + verificationDate;  }    public void destroy() throws Exception {  System.out.println("VoterVerifier.destroy()");  name = null;  age = 0;  verificationDate = null;    }    public void afterPropertiesSet() throws Exception {  System.out.println("VoterVerifier.afterPropertiesSet()");    // Initialization the value which are not taking particapting in dependency  // injection  verificationDate = new Date();    // validation  if (name == null || age <= 0) {  throw new IllegalArgumentException("Invalid User Input");  }    }    } |
| ***ApplicationContex.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 http://www.springframework.org/schema/beans/spring-beans.xsd">    <!-- spring bean configuration -->  <bean id="voter" class="com.ab.springlifecycle.beans.VoterVerifier">  <property name="name" value="raja"></property>  <property name="age" value="19"></property>  </bean>    </beans> |
| Client App:  package com.ab.springlifecycle;    import org.springframework.context.support.ClassPathXmlApplicationContext;    import com.ab.springlifecycle.beans.VoterVerifier;    public class App  {  public static void main( String[] args ){  // create IOC container:  ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("com/ab/springlifecycle/cfg/applicationContext.xml");    // get spring bean object:  VoterVerifier voter = ctx.getBean("voter", VoterVerifier.class);    // invokes business method:  try {  System.out.println(voter.checkVotingEligibility());  } catch (Exception e) {  e.printStackTrace();  }    // close the container  ctx.close();  }  } |

Pros of Programmatic approach:

* No need of configuring spring bean life cycle method explicitly by seeing special interface implementation the IOC container call **afterPropertySet()**, **destroy()** methods as the life cycle method
* Here spring bean life cycle methods names are fixed names.

Cons of Programmatic approach:

* Makes spring bean as invasive classes.
* Cannot be used while configuring 3rd party supplied, class, other than spring api supplied classes, user-defined classes as spring beans.

***Annotation Approach:***

* We generally use this technique while developing spring application in:
  + Annotation Driven Configuration
  + 100% Code Driven Configuration
  + Spring Boot Configuration
* For example go to the link: [abhisheksharmas04/Spring-Bean-LifeCyecl (github.com)](https://github.com/abhisheksharmas04/Spring-Bean-LifeCyecl)

Pros of Annotation approach:

* Can be used in annotation driven configuration, 100% code driven configuration and in spring boot.
* We can minimize xml code based configurations.

Cons of Annotation approach:

* We can use only in user defined classes that are configured as spring beans.
* From java 9 separate jar file needs to be added (javax.annotation-api <version>.jar)

On a spring bean if we enable spring bean life cycle management in all three approaches in which order they will be executed?

* First annotation configuration --> Programmatic --> Declarative

## Conclusion on Spring bean life cycle

* If spring bean class is user defined class and the spring application is developed in annotation configuration or 100% code configuration or spring boot then go for Annotation based spring bean life cycle.
* If spring bean class is the spring api supplied class and if it is implementing **InitializingBean & DisposableBean** then go for programmatic approach of spring bean life cycle otherwise go for decelerative approach of spring bean life cycle.
* If spring bean class is Third party supplied class, then go for decelerative approach of spring bean life cycle.
* If spring bean class is user-defined class of XML driven configuration then go for decelerative approach.