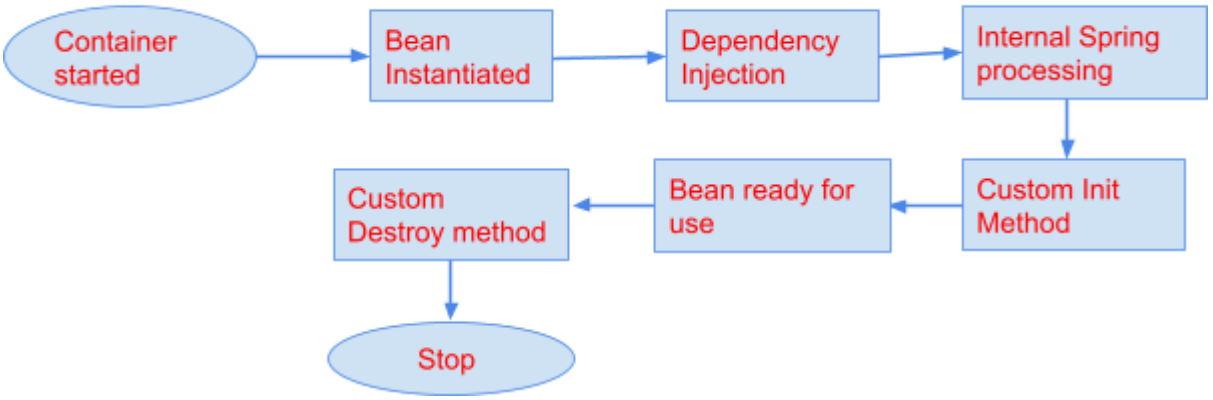
Lifecycle of a Bean



- Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started and the application context is loaded.
- After that, the container creates the instance of a bean as per the request.
- In dependency injection phase the values for the bean attributes gets created using constructor or setter injection
- After dependency injection spring pre processing will happen to keep track of everything and if you want to connect to database the logic related to that can be written inside init()
- Now bean is ready for use
- 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.

Custom init() and destroy():

In Spring, you can use init-method and destroy-method as attribute in bean configuration file for bean to perform certain actions upon initialization and destruction

Scenario:
Now let's try to print a bean created while bean instantiated and bean destroyed once it gets destroyed.

Step 1: Insert myStartup() and myClosing() in Audi class as shown below

```
void MystartUp(){
	System.out.println("Bean Created");
}

void MyClosing(){
	System.out.println("Bean Destroyed");
}
```

Audi.java

```
package com.tapacad.spring;

public class Audi implements Car
{
	private RocketEngine rocketEngine;
	private String colour;
	private int price;

	@Override
	public String getCarDetails() {
		return "Audi";
	}

	@Override
	public String getEngineDetail() {
		return rocketEngine.getEngineDetail();
	}

	public RocketEngine getRocketEngine() {
```

```
            return rocketEngine;
      }

      public void setRocketEngine(RocketEngine rocketEngine) {
            this.rocketEngine = rocketEngine;
      }

      public String getColour() {
            return colour;
      }

      public void setColour(String colour) {
            this.colour = colour;
      }

      public int getPrice() {
            return price;
      }

      public void setPrice(int price) {
            this.price = price;
      }

      void MystartUp(){
            System.out.println("Bean Created");
      }

      void MyClosing(){
            System.out.println("Bean Destroyed");
      }

}
```

Step 2: insert init-method and destroy-method is the attribute of spring <bean> tag in applicationContext.

init-method is used to declare a custom method for the bean which will act as bean initialization method.

destroy-method is a bean attribute using which we can assign a custom bean method that will be called just before the bean is destroyed.

```
<bean id ="audi" class="com.tapacad.spring.Audi"
                        init-method="myStartUp"
```

```
                    destroy-method="myClosing">
     <property name="rocketEngine" ref = "engine"></property>
     <property name="colour" value = "${colour}"></property>
     <property name="price" value = "${price}"></property>
</bean>
```

Step 3:

```
package com.tapacad.spring;

import
org.springframework.context.support.ClassPathXmlApplicationContext
;

public class MyApp6 {

     public static void main(String[] args) {
//          Load Application context
          ClassPathXmlApplicationContext context =
                    new
ClassPathXmlApplicationContext("applicationContext.xml");

//          Get bean
          Car car = (Car)context.getBean("audi");

//          print object reference of car
          System.out.println(car);

//          close context
          context.close();

     }

}
```

Output:

```
Bean Created
com.tapacad.spring.Audi@6f03482
Bean Destroyed
```

If you observe from the above output myStartUp() executes when bean is instantiated and myDestroy() before bean gets destroyed. **Here the scope of bean is singleton**

Now let's see how myStartUp() and myDestroy() will work when the scope is prototype

MyApp6.java

```
package com.tapacad.spring;

import
org.springframework.context.support.ClassPathXmlApplicationContext
;

public class MyApp6 {

	public static void main(String[] args) {
//		Load Application context
		ClassPathXmlApplicationContext context =
				new
ClassPathXmlApplicationContext("applicationContext.xml");

//		Get bean
		Car car = (Car)context.getBean("audi");

//		print object reference of car
		System.out.println(car);

//		close context
		context.close();

	}
```

```
}
```

Output:

```
Bean Created
com.tapacad.spring.Audi@179ece50
```

If you observe from the above output in prototype scope **display() is not executed**

Till now we have created an init and destroy method inside the audi. If in case there are 100 bean objects you need to create a 100 times init and destroy method, to avoid this situation you can include this inside beans tag and mention default method over there.

If you have not created init and destroy method inside the specific bean then fun1 and fun2 method inside the beans tag will execute

```
default-init-method="fun1"
default-destroy-method="fun2"
```

applicationContext.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"

        default-init-method="fun1"
        default-destroy-method="fun2"
```

```
        >
        <context:property-placeholder
location="classpath:car.properties"/>



        <!-- bean definitions here -->

           <bean id="bmw" class="com.tapacad.spring.BMW">
                <constructor-arg ref ="engine"></constructor-arg>
           </bean>

           <bean id="nano" class="com.tapacad.spring.Nano">
                <constructor-arg ref ="engine"></constructor-arg>
           </bean>

           <bean id ="audi" class="com.tapacad.spring.Audi" >
                <property name="rocketEngine" ref =
"engine"></property>
                <property name="colour" value =
"${colour}"></property>
                <property name="price" value =
"${price}"></property>
           </bean>

           <bean id ="engine"
class="com.tapacad.spring.RocketEngine">
           </bean>
</beans>
```

- Create fun1() and fun2() inside audi class

```
void fun1(){
     System.out.println("Default Bean Created");
}

void fun2(){
     System.out.println("Default Bean Destroyed");
}
```

Audi.java

```
package com.tapacad.spring;

public class Audi implements Car
{
	private RocketEngine rocketEngine;
	private String colour;
	private int price;

	@Override
	public String getCarDetails() {
		return "Audi";
	}

	@Override
	public String getEngineDetail() {
		return rocketEngine.getEngineDetail();
	}

	public RocketEngine getRocketEngine() {
		return rocketEngine;
	}

	public void setRocketEngine(RocketEngine rocketEngine) {
		this.rocketEngine = rocketEngine;
	}

	public String getColour() {
		return colour;
	}

	public void setColour(String colour) {
		this.colour = colour;
	}

	public int getPrice() {
		return price;
	}

	public void setPrice(int price) {
		this.price = price;
```

```
	}

	void myStartUp(){
		System.out.println("Bean Created");
	}

	void myClosing(){
		System.out.println("Bean Destroyed");
	}

	void fun1(){
		System.out.println("Default Bean Created");
	}

	void fun2(){
		System.out.println("Default Bean Destroyed");
	}



}
```

MyApp6.java

```
package com.tapacad.spring;

import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Set;

import
org.springframework.context.support.ClassPathXmlApplicationContext
;

public class MyApp6 {

	public static void main(String[] args) {
//		Load Application context
		ClassPathXmlApplicationContext context =
```

```
                        new
ClassPathXmlApplicationContext("applicationContext.xml");

//          Get bean
            Car car = (Car)context.getBean("audi");

//          print object reference of car
            System.out.println(car);

//          close context
            context.close();
      }

}
```

Output:

```
Default Bean Created
com.tapacad.spring.Audi@6f03482
Default Bean Destroyed
```

If you observe from the above output default method that is fun1() and fun2() is called

Note:

If in case along with default init and destroy method if you specified the init and destroy method in bean that time the methods specified in the eban will get executed.