**6. Spring Bean Scopes and Lifecycle**

**Bean in Spring**:

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.

**Bean Scopes**:

The scope of a bean defines the life cycle and visibility of that bean in the contexts in which it is used.

The latest version of Spring framework defines 6 types of scopes:

1. singleton
2. prototype
3. request
4. session
5. application
6. websocket

The last four scopes mentioned request, session, application and websocket are only available in a web-aware application.

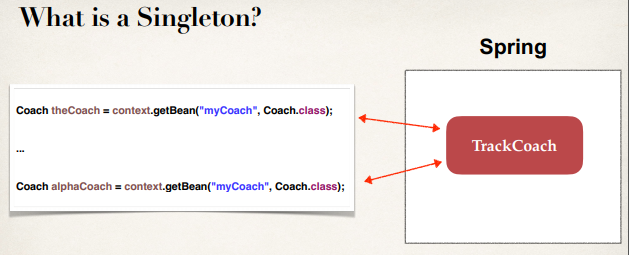
**Default scope**:

The default scope of a bean is singleton.

**1) Singleton**:

1. For singleton Spring container creates only one instance of the bean, by default.
2. It cached in memory
3. All request of the bean will return a shared reference to the same bean.

So the end result is that there is only one bean and everyone will share it.



Here we have two object references here and they point to the same area of memory.

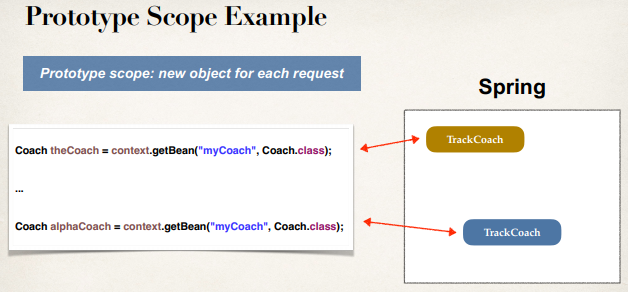
Or they point to the same bean.

In spring the default scope of a bean is singleton but we can explicitly specify this.



**2) Prototype**:

In prototype scope a new object is created for each request.



**Example**:

**Coach.java (Interface)**:

**package** com.odduu.ruhul;

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

**public** String getDailyWorkout();

**public** String getDailyFortune();

}

**TrackCoach.java (Class)**:

**package** com.odduu.ruhul;

**public** **class** TrackCoach **implements** Coach {

**private** FortuneService fortuneService;

**public** TrackCoach(FortuneService fortuneService) {

**this**.fortuneService = fortuneService;

}

@Override

**public** String getDailyWorkout() {

**return** "Run a hard 5k";

}

@Override

**public** String getDailyFortune() {

**return** "Just Do It: "+fortuneService.getFortune();

}

}

**FortuneService.java (Interface)**:

**package** com.odduu.ruhul;

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

**public** String getFortune();

}

**HappyFortuneService.java (class)**:

**package** com.odduu.ruhul;

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

@Override

**public** String getFortune() {

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

}

}

**beanScope-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"*>

<!-- Define your beans here -->

<!-- define the dependency -->

<bean id=*"myFortune"* class=*"com.odduu.ruhul.HappyFortuneService"*>

</bean>

<bean id=*"myCoach"*

class=*"com.odduu.ruhul.TrackCoach"*

scope=*"singleton"*>

<!-- set up constructor injection -->

<constructor-arg ref=*"myFortune"* />

</bean>

</beans>

**BeanScopeDemoApp.java (Class)**:

**package** com.odduu.ruhul;

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

**public** **class** BeanScopeDemoApp {

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

// load the spring configuration file

ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("beanScope-applicationContext.xml");

// retrieve bean from container

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

Coach alphaCoach = context.getBean("myCoach", Coach.**class**);

// check if they are the same

**boolean** result = (theCoach == alphaCoach);

// print out the result

System.***out***.println("Pointing to the same object: " + result);

System.***out***.println("Memory location for theCoach: " + theCoach);

System.***out***.println("Memory location for alphaCoach: " + alphaCoach);

// close the context

context.close();

}

}

/\*

Pointing to the same object: true

Memory location for theCoach: com.odduu.ruhul.TrackCoach@6737fd8f

Memory location for alphaCoach: com.odduu.ruhul.TrackCoach@6737fd8f

\*/

If we change the scope from *singleton* to *prototype* then we will get the following output.

/\*

Pointing to the same object: false

Memory location for theCoach: com.odduu.ruhul.TrackCoach@30b8a058

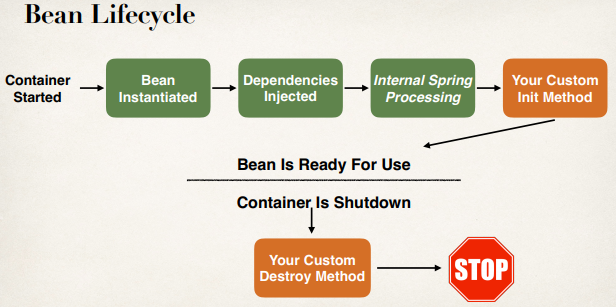
Memory location for alphaCoach: com.odduu.ruhul.TrackCoach@7494e528

\*/

**Bean Lifecycle**:

When the spring container fast starts there are a couple of things that happens.

1. Beans are instantiated
2. Dependences are injected
3. Internal spring processing
4. Our custom Init method
5. Bean is ready for use and container is shutdown
6. Our custom destroy method
7. Container shutdown. (context.close())



**Bean Lifecycle method**:

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

**Init-method configuration**:

<beans … >

<bean id="myCoach"

class="com.luv2code.springdemo.TrackCoach"

init-method="doMyStartupStuff">

…

</bean>

</beans>

**Destroy-method configuration**:

<beans … >

<bean id="myCoach"

class="com.luv2code.springdemo.TrackCoach"

init-method="doMyStartupStuff"

destroy-method="doMyCleanupStuff">

…

</bean>

</beans>

**Development Process**:

1. Define your methods for init and destroy
2. Configure the method names in Spring config file

**Init-method**:

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

System.***out***.println("TrackCoach: inside method doMyStartupStuff()");

}

**Destroy-method**:

**public** **void** doMyCleaneupStuff() {

System.***out***.println("TrackCoach: inside method doMyCleaneupStuff()");

}

**Example**:

Here the above example code is modify in some places. We add init and destroy method in our "TrackCoach.java" class.

**TrackCoach.java (class)**:

**package** com.odduu.ruhul;

**public** **class** TrackCoach **implements** Coach {

**private** FortuneService fortuneService;

**public** TrackCoach(FortuneService fortuneService) {

**this**.fortuneService = fortuneService;

}

@Override

**public** String getDailyWorkout() {

**return** "Run a hard 5k";

}

@Override

**public** String getDailyFortune() {

**return** "Just Do It: " + fortuneService.getFortune();

}

// add an init method

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

System.***out***.println("TrackCoach: inside method doMyStartupStuff()");

}

// add an destroy method

**public** **void** doMyCleaneupStuff() {

System.***out***.println("TrackCoach: inside method doMyCleaneupStuff()");

}

}

Add an XML file name "beanLifeCycle-applicationContext.xml"

**beanLifeCycle-applicationContext.xml (configuration 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"*

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"*>

<!-- Define your beans here -->

<!-- define the dependency -->

<bean id=*"myFortune"* class=*"com.odduu.ruhul.HappyFortuneService"*>

</bean>

<bean id=*"myCoach"* class=*"com.odduu.ruhul.TrackCoach"*

init-method=*"doMyStartupStuff"*

destroy-method=*"doMyCleaneupStuff"*>

<!-- set up constructor injection -->

<constructor-arg ref=*"myFortune"* />

</bean>

</beans>

6. Spring Bean Scopes and Lifecycle