Bean Scopes

Bean Scopes

- Scope refers to the lifecycle of a bean
- How long does the bean live?
- How many instances are created?
- How is the bean shared?

Default Scope: Singleton

```
<bean id="myCoach"
    class="com.springdemo.TrackCoach">
 </bean>
```

What Is a Singleton?

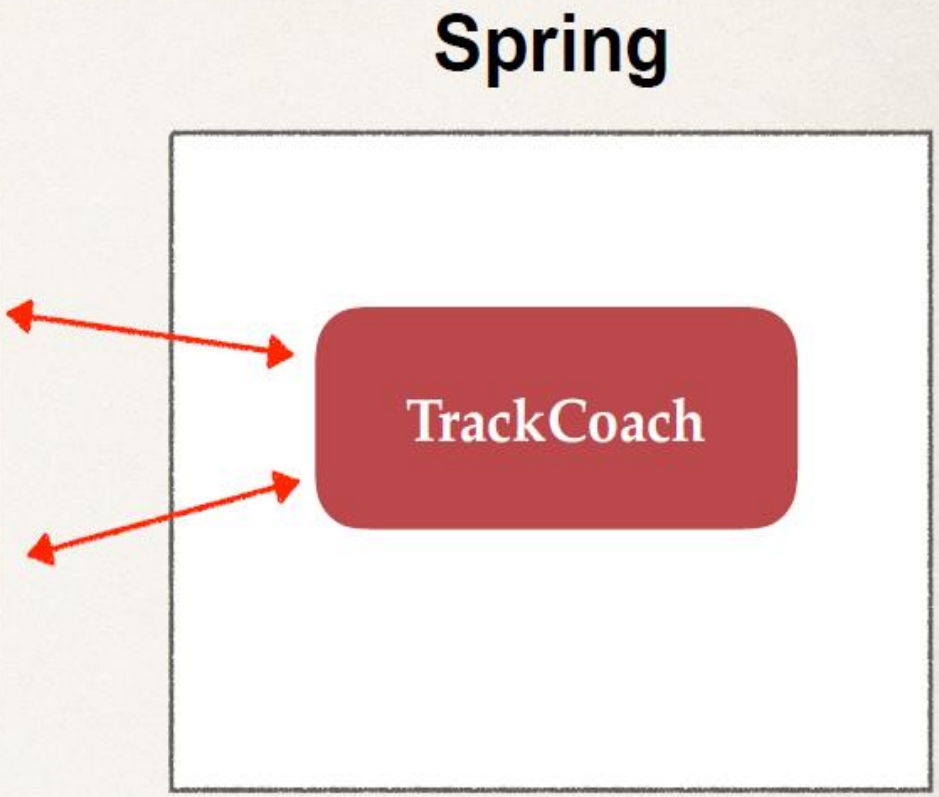
- Spring Container creates only one instance of the bean, by default
- It is cached in memory
- All requests for the bean
  - will return a SHARED reference to the SAME bean

What Is a Singleton?

- Spring Container creates only one instance of the bean, by default
- It is cached in memory
- All requests for the bean
  - will return a SHARED reference to the SAME bean

What is a Singleton?

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

...

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



Explicitly Specify Bean Scope

```
<beans ... >


    <bean id="myCoach"
        class="com.springdemo.TrackCoach"
        scope="singleton">

        <!-- set up constructor injection -->
        <constructor-arg ref="myFortuneService" />
    </bean>


</beans>
```

Additional Spring Bean Scopes

Scope	Description
singleton	Create a single shared instance of the bean. Default scope.
prototype	Creates a new bean instance for each container request.
request	Scoped to an HTTP web request. Only used for web apps.
session	Scoped to an HTTP web session. Only used for web apps.
global-session	Scoped to a global HTTP web session. Only used for web apps.

Prototype Scope Example

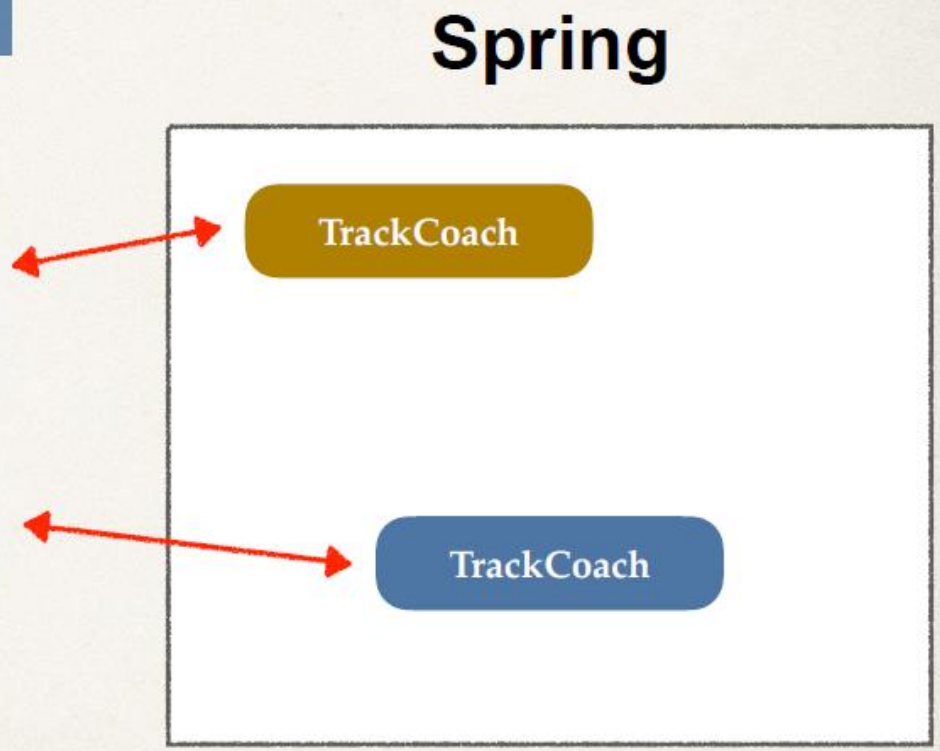
Prototype scope: new object for each request

```
<beans ... >


    <bean id="myCoach"
        class="com.luv2code.springdemo.TrackCoach"
        scope="prototype">


</beans>
```

Prototype Scope Example

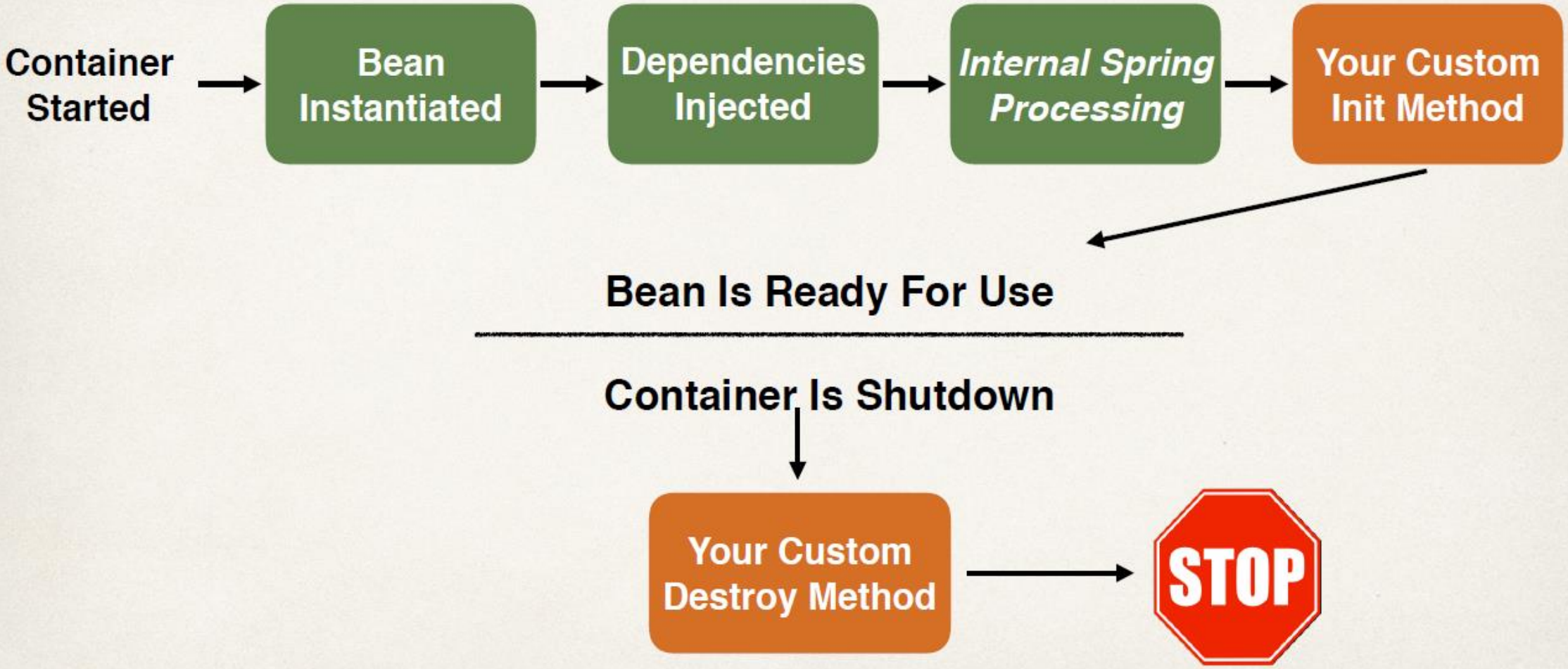
Prototype scope: new object for each request

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

...

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



Bean Lifecycle Methods

Bean Lifecycle



Bean Lifecycle Methods / Hooks

- You can add custom code during **bean initialization**
  - Calling custom business logic methods
  - Setting up handles to resources (db, sockets, file etc)

- You 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.springdemo.TrackCoach"
        init-method="doMyStartupStuff"
        destroy-method="doMyCleanupStuffYoYo">

        <!-- set up constructor injection -->
        <constructor-arg ref="myFortuneService" />
    </bean>

</beans>
```

Destroy: method configuration

```
<beans ... >


  <bean id="myCoach"
      class="com.springdemo.TrackCoach"
      init-method="doMyStartupStuff"
      destroy-method="doMyCleanupStuffYoYo">

      <!-- set up constructor injection -->
      <constructor-arg ref="myFortuneService" />
  </bean>


</beans>
```

Development Process

Step-By-Step

1. Define your methods for init and destroy

2. Configure the method names in Spring config file

Spring Configuration with Annotations

What are Java Annotations?

- Special labels/markers added to Java classes
- Provide meta-data about the class
- Processed at compile time or run-time for special processing

Annotation Example

- We've seen annotations already …

```
public class TrackCoach implements Coach {

  @Override
  public String getDailyWorkout() {
    return "Run a hard 5k";
  }
  ...
}
```

Why Spring Configuration with Annotations?

- XML configuration can be verbose
- Configure your Spring beans with Annotations
- Annotations minimizes the XML configuration

Scanning for Component Classes

- Spring will scan your Java classes for special annotations
- Automatically register the beans in the Spring container

Development Process

Step-By-Step

1. Enable component scanning in Spring config file
2. Add the @Component Annotation to your Java classes
3. Retrieve bean from Spring container

Step 1: Enable component scanning in Spring config file

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

<context:component-scan base-package="com.springdemo" />
```

Step 2: Add the @Component Annotation to your Java classes

```
@Component("thatSillyCoach")
public class TennisCoach implements Coach {

  @Override
  public String getDailyWorkout() {
    return "Practice your backhand volley";
  }

}
```

Step 3: Retrieve bean from Spring container

- Same coding as before … nothing changes.

```
Coach theCoach = context.getBean("thatSillyCoach", Coach.class);
```