

## 40.Introduction to Beans scope inside Spring

Bean scope is a concept inside spring which will define how a bean has to be created and maintained by spring IOC container, whether it needs to be maintained forever inside an application till the restart happens, or whether it needs to create a new bean every time a reference is asked.

So all these are controlled with the help of bean scopes inside spring

Request, Session and Application are web application scopes.

## 41.Deep dive on Singleton Bean Scope:

easy bytes

### SINGLETON BEAN SCOPE

- Singleton is the default scope of a bean in Spring. In this scope, for a single bean we always get a same instance when you refer or autowire inside your application.
- Unlike Singleton design pattern where we have only 1 instance in entire app, inside Singleton scope Spring will make sure to have only 1 instance per unique bean. For example, if you have multiple beans of same type, then Spring Singleton scope will maintain 1 instance per each bean declared of same type.

```
@Component
@Scope(BeanDefinition.SCOPE_SINGLETON)
public class VehicleServices {

    VehicleServices vehicleServices1 = context.
        getBean(VehicleServices.class);
    VehicleServices vehicleServices2 = context.
        getBean(name: "vehicleServices", VehicleServices.class);
}
```

Creates a single bean inside Spring context

The two variables refers to the same bean inside Spring context

By default if we don't specify any scope Singleton will be considered as default scope.

```
1 package com.chinmay.beans;
2
3 import org.springframework.beans.factory.config.BeanDef
4 import org.springframework.context.annotation.Scope;
5 import org.springframework.stereotype.Component;
6
7 @Component("VehicleServices") 5 usages
8 @Scope(BeanDefinition.SCOPE_SINGLETON)
9
10 public class Vehicle {
11
12     String name; 2 usages
13
14     public String getName() { no usages
15         return name;
16     }
17
18     public void setName(String name) { no usages
19         this.name = name;
20     }
21 }
22
23
```

We could see here as the Scope is Single ton Same instance of the bean has been returned both times when we have fetched the Bean from the context.

```
1 package com.chinmay.main;
2
3 import com.chinmay.beans.Vehicle;
4 import com.chinmay.config.ProjectConfig;
5 import org.springframework.context.ApplicationContext;
6 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
7
8 public class Ex_17_SingleTon_Bean_Scope {
9     public static void main(String[] args) {
10         ApplicationContext context=new AnnotationConfigApplicationContext(ProjectConfig.class);
11         Vehicle veh=context.getBean(Vehicle.class);
12         Vehicle veh1=context.getBean(name: "VehicleServices", Vehicle.class);
13         System.out.println("Hash Code of Object 1: "+veh.hashCode());
14         System.out.println("Hash Code of Object 2 :"+veh1.hashCode());
15
16         if(veh==veh1)
17         {
18             System.out.println("Same bean is returned from the context");
19         }
20     }
21 }
22
23
```

C:\Program Files\Java\jdk-17\bin\java.exe ...

Hash Code of Object 1: 1051876890

Hash Code of Object 2 :1051876890

Same bean is returned from the context

## 42.Race Condition:

eazy bytes

### RACE CONDITION

- A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable. Then the first thread and second thread perform their operations on the value, and they race to see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.

```
// Shared Value inside a object
Map<String, String> reservedTables = new HashMap<>();

// thread -1
if (!reservedTables.containsKey("table1")){
    reservedTables.put("table1", "USER1");
}

// thread - 2
if (!reservedTables.containsKey("table1")){
    reservedTables.put("table1", "USER2");
}
```

→

Thread 1




table1 -> USER1

Thread 2




table1 -> USER2

If both threads run the if condition at a same time, then both of them will reserve the table for different users

Since the same instance of the bean will be returned every time when we try to fetch the bean .There might be chance that two different beans will try to update the bean details and same time and overwriting of the data takes place.

## 43.Use cases of Singleton bean scope

eazy bytes

### USE CASES OF SINGLETON BEANS

Singleton Beans use cases

- ✓ Since the same instance of singleton bean will be used by multiple threads inside your application, it is very important that these beans are immutable.
- ✓ This scope is more suitable for beans which handles service layer, repository layer business logics.

- 1

Building mutable singleton beans, will result in the race conditions inside multi thread environments.
- 2

There are ways to avoid race conditions due to mutable singleton beans with the help of synchronization.
- 3

But it is not recommended, since it brings lot of complexity and performance issues inside your app. So please don't try to build mutable singleton beans.

#### 44.Eager and Lazy instantiation of Singleton scope.

We can control when the Singleton beans will be created by making the bean as Lazy instantiation. So that bean will not be created until there is need for that.

Most of the times we will go with the Eager instantiation.

But we change the bean instantiation to Lazy instantiation when bean is used very remotely or very rarely.

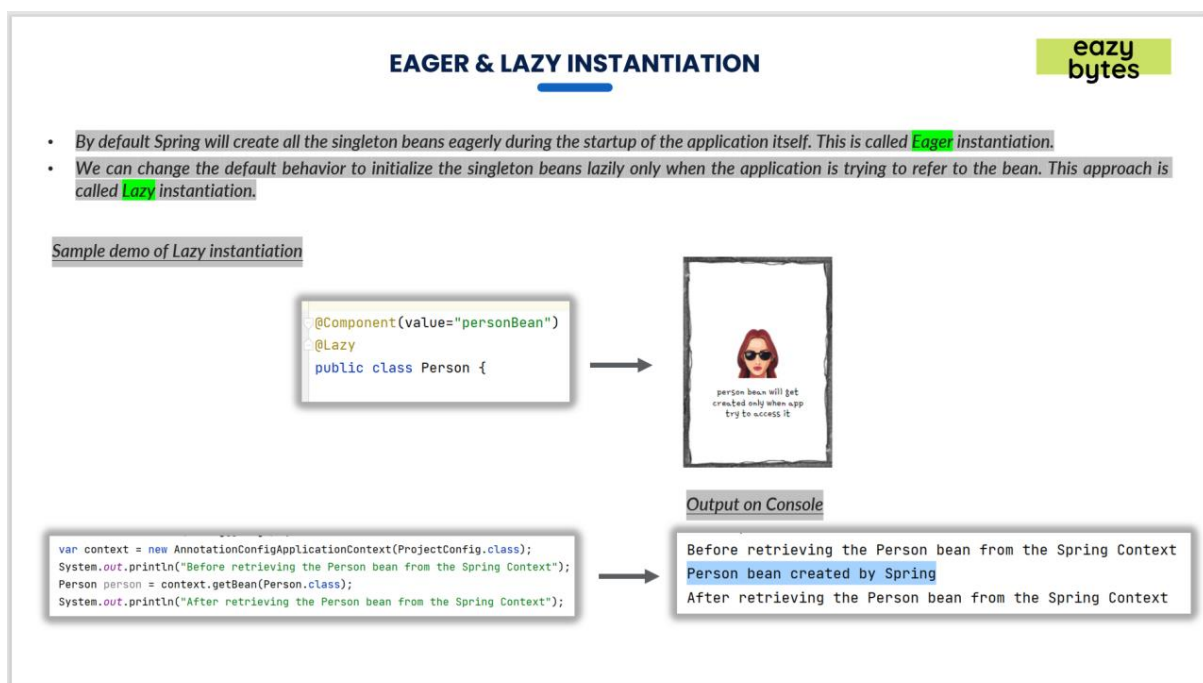
Ex: Deletion of Account

There might be chance these beans might not be used so creating them during the startup will not have much use and increase the number of the beans in the context.

Issue with Lazy instantiation:

In case of Lazy instantiation an exception will be raised during the run time if there is an issue while creating the bean.

Whereas if go with Eager instantiation the server with not startup if there is an issue while creating the bean.



Bean will get created when it is referred for the first time rather than during the startup of Application.

## 45. Demo of Eager and lazy instantiation of Singleton bean

Ex:18

Ex\_18\_Eazy\_Lazy\_Instantiation

```
package com.chinmay.beans;

import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

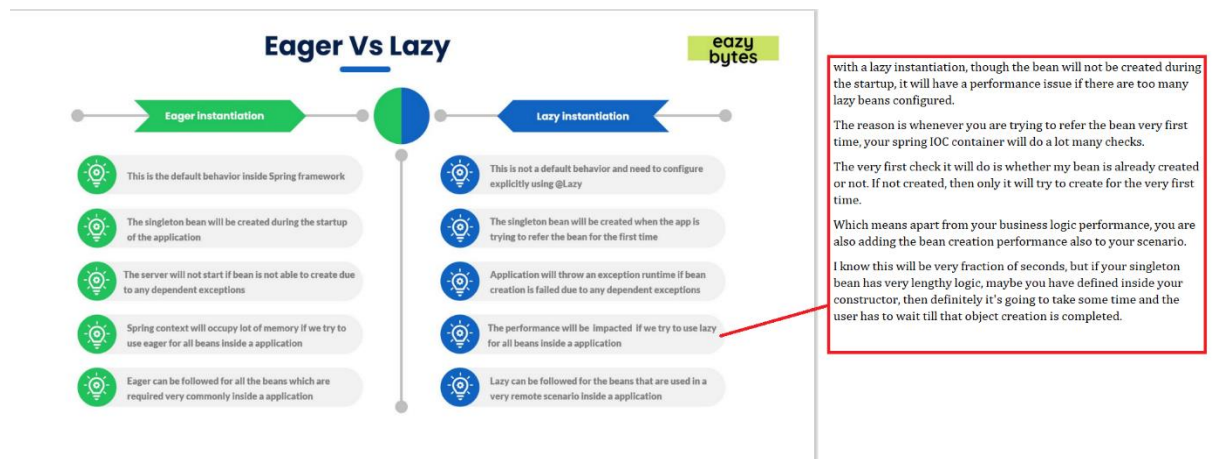
@Component(value="personbean")
@Lazy
public class Person {
    public Person() {
        System.out.println("Person bean is created");
    }
}

import com.chinmay.beans.Person;
import com.chinmay.beans.Vehicle;
import com.chinmay.config.ProjectConfig;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Ex_18_Eazy_Lazy_Instantiation {
    public static void main(String[] args) {
        ApplicationContext context=new AnnotationConfigApplicationContext(P
        System.out.println("Above are the Beans created during the startup");
        Vehicle vehicle=context.getBean( name: "VehicleBean", Vehicle.class);
        System.out.println("Beans Created after the startup : ");
        Person person=context.getBean( name: "personbean", Person.class);
    }
}
```

As the person is set as Lazy bean due to which we could see here that it has been created during its first usage rather than during the start of the Application

## 46. Eager Initialization vs Lazy Initialization



47: Deep dive of prototype Bean scope:

If there is an dependency of Prototype Scope bean inside the Singleton Scope bean how many we call it will return the same instance of bean even though it is of Proto type scope as we are accessing it from Singleton Scope.

```

@Component("vehicleBean")
public class Vehicle {

    private String name="Honda";
    private final VehicleService vehicleService;

    @Autowired
    public Vehicle(VehicleService vehicleService){
        this.vehicleService = vehicleService;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public VehicleService getVehicleServices() {
        return vehicleService;
    }
}

```

Here the Vehicle is of Single ton scope where as the vehicle is dependent on VehicleServiceBean which is of prototype Scope .  
Even though it is of Prototype scope as it is used in Single ton scope bean every time we request for VehicleServiceBean same instance of the copy will be returned.

## PROTOTYPE BEAN SCOPE

easy  
bytes

- With prototype scope, every time we request a reference of a bean, Spring will create a new object instance and provide the same.
- Prototype scope is rarely used inside the applications and we can use this scope only in the scenarios where your bean will frequently change the state of the data which will result race conditions inside multi thread environment. Using prototype scope will not create any race conditions.

```

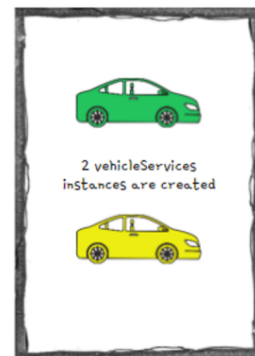
@Component
@Scope(BeaDefinition.SCOPE_PROTOTYPE)
public class VehicleServices {

```

```

VehicleServices vehicleServices1 = context.
    getBean(VehicleServices.class);
VehicleServices vehicleServices2 = context.
    getBean( name: "vehicleServices", VehicleServices.class);

```



```

package com.chinmay.beans;

import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component("vehicleBean")
@Scope(BeaDefinition.SCOPE_PROTOTYPE)
public class Vehicle {

    public Vehicle() { System.out.println("Vehicle is created"); }
}

```

```

pom.xml (Ex_19_Proto_Type_Scope)  Ex_19_Proto_Type_Scope.java  Vehicle.java  ProjectConfig.java

1 package com.chinmay.main;
2
3 > import ...
4
5
6
7
8
9 public class Ex_19_Proto_Type_Scope {
10     public static void main(String[] args) {
11
12         ApplicationContext context=new AnnotationConfigApplicationContext(ProjectConfig.class);
13         System.out.println("Above are the Beans created during the startup : \n");
14         Vehicle vehicle=context.getBean( name: "vehicleBean", Vehicle.class);
15         Vehicle vehicle1=context.getBean( name: "vehicleBean", Vehicle.class);
16
17         if(vehicle1==vehicle){
18             System.out.println("Beans created are of singleton Scope");
19         }
20         else {
21             System.out.println("Beans created are of prototype Scope");
22         }
23     }
24 }

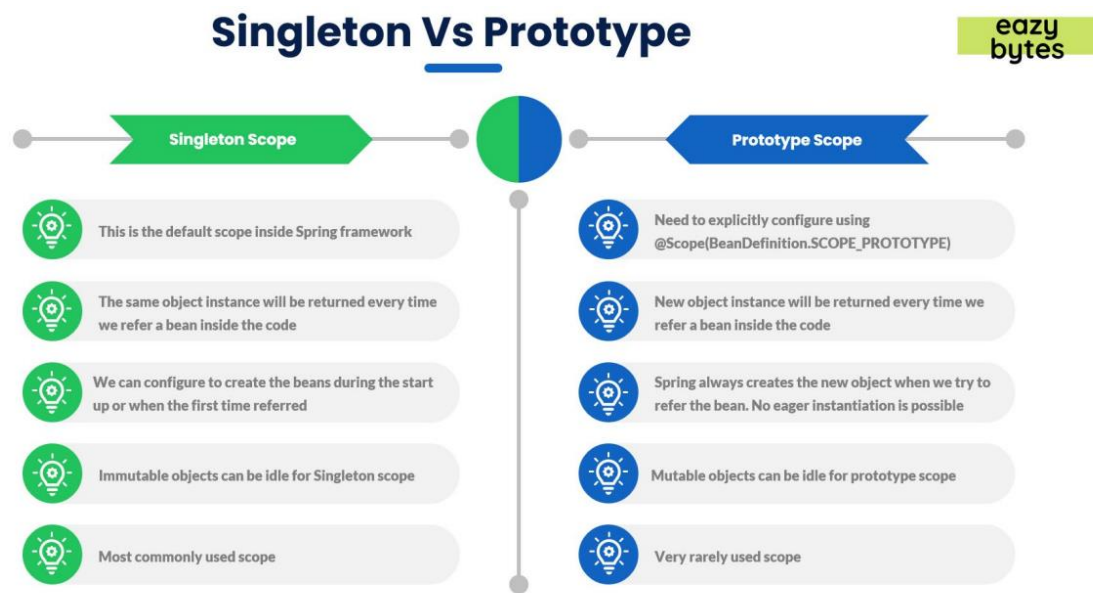
```

```

C:\Program Files\Java\jdk-17\bin\java.exe ...
Person bean is created
Above are the Beans created during the startup :
Vehicle is created
Vehicle is created
Beans created are of prototype Scope
Process finished with exit code 0

```

## 49.Singleton vs Prototype Scope



In Prototype Scope we don't have lazy or Eager initialization.