BEAN SCOPES – PROTOTYPE AND SINGLETON

SINGLETON BEAN:

In Spring Framework, the Singleton scope is a common bean scope used to define a bean that is instantiated only once per Spring IoC container. This means that, regardless of how many times the bean is requested from the container, the same instance of the bean is returned every time.

By default, all beans defined in the Spring context are singleton beans. The singleton scope is useful for beans that maintain state and should be shared across the entire application. Examples of singleton beans include database connections, thread pools, and other shared resources.

PROTOTYPE BEAN:

In Spring Framework, the Prototype scope is a bean scope used to define a bean that is instantiated every time it is requested from the Spring IoC container. This means that a new instance of the bean is created every time the bean is injected or retrieved from the container.

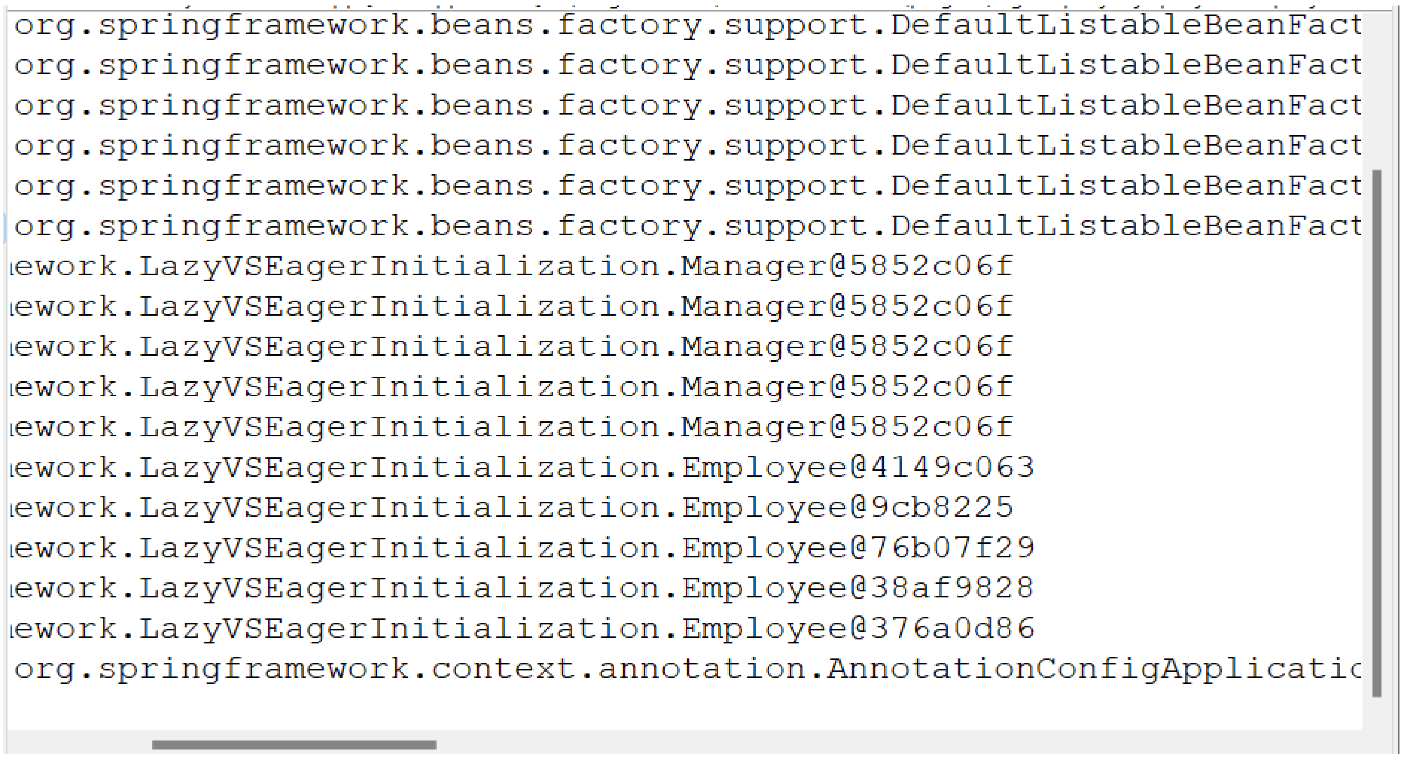
Prototype scoped beans are useful for objects that maintain state that is specific to a single use or request, such as a user session in a web application. Each time the user initiates a new session, a new instance of the bean is created.

1. Java Singleton (GOF) vs Spring Singleton
    a. Spring Singleton: One object instance per Spring IoC container
    b. Java Singleton (GOF): One object instance per JVM

EXAMPLE:

```
package com.naveen.learnspringframework.LazyVSEagerInitialization;

import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
class Manager{

}

@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
@Component
class Employee{

}

@Configuration
@ComponentScan
public class LazyInitializationApp {

    public static void main(String[] args) {

        try(var context =
                new AnnotationConfigApplicationContext
                    (LazyInitializationApp.class)){

            System.out.println(context.getBean(Manager.class));
            System.out.println(context.getBean(Manager.class));
            System.out.println(context.getBean(Manager.class));
            System.out.println(context.getBean(Manager.class));
            System.out.println(context.getBean(Manager.class));

            System.out.println(context.getBean(Employee.class));
            System.out.println(context.getBean(Employee.class));
            System.out.println(context.getBean(Employee.class));
            System.out.println(context.getBean(Employee.class));
            System.out.println(context.getBean(Employee.class));
        }
    }
}
```

OUTPUT:



```
org.springframework.beans.factory.support.DefaultListableBeanFact
org.springframework.beans.factory.support.DefaultListableBeanFact
org.springframework.beans.factory.support.DefaultListableBeanFact
org.springframework.beans.factory.support.DefaultListableBeanFact
org.springframework.beans.factory.support.DefaultListableBeanFact
org.springframework.beans.factory.support.DefaultListableBeanFact
ework.LazyVSEagerInitialization.Manager@5852c06f
ework.LazyVSEagerInitialization.Manager@5852c06f
ework.LazyVSEagerInitialization.Manager@5852c06f
ework.LazyVSEagerInitialization.Manager@5852c06f
ework.LazyVSEagerInitialization.Manager@5852c06f
ework.LazyVSEagerInitialization.Employee@4149c063
ework.LazyVSEagerInitialization.Employee@9cb8225
ework.LazyVSEagerInitialization.Employee@76b07f29
ework.LazyVSEagerInitialization.Employee@38af9828
ework.LazyVSEagerInitialization.Employee@376a0d86
org.springframework.context.annotation.AnnotationConfigApplicatic
```

In this above output, note the address of Manager bean and Employee bean.