Singleton Bean Scope

Spring Framework Core

Discover here the main character among scopes that exists in beans within Spring Framework: **singleton bean scope**.

(Includes code where I uploaded to my github repo. Link in the description of this post)

@mauricioperez

Bean scopes in Spring

1. Singleton

2. **Prototype**
3. **Request**
4. **Session**
5. **Application**

In this post we are seeing the number 1: **Singleton.**

Singleton Scope also **is the default option** when declaring beans.

02 What is Singleton Scope?

Singleton Scope is a way of telling our Spring Context **how it is going to save and create instances** for future usages in our application.

Singleton's way of operating a bean is to create **one and only one instance** of a defined bean (also defined as a recipe).

03

Uniqueness of Singleton Scope

Here's the thing:

No matter how many calls we do to the bean, **we always be getting the same instance** injected through Spring Context.

This gives me consistency of my bean since this instances are inmutable.

04

Use cases for Singleton Scope

This scope is more suitable for beans which handles **service layers, repository business layers logics**.

This represents an advantage in the case we require a **fixed logic from our service layer** and doesn’t neccessarily need to create replicas of it.

Take a look at

some code

I prepared for you!

Situation of the example

05

The following short example tries to show instantiations of the bean using Singleton Scope aboarding a car models creation.

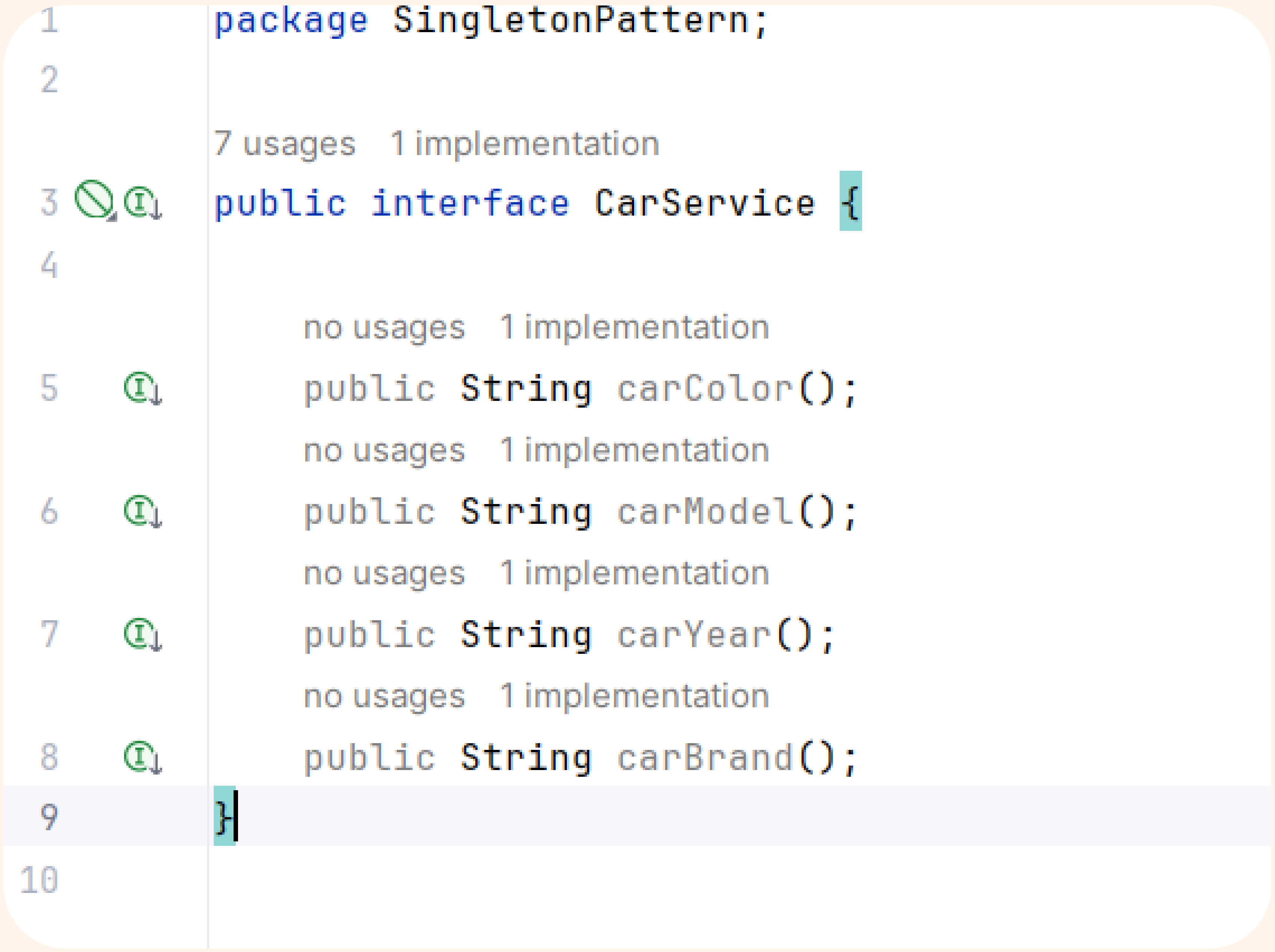
In this case we are creating a Toyota implementation car and we'll see the effects of Singleton Scope.

Remember this code is on my github profile

Let's see the parts...

Interface creation

06

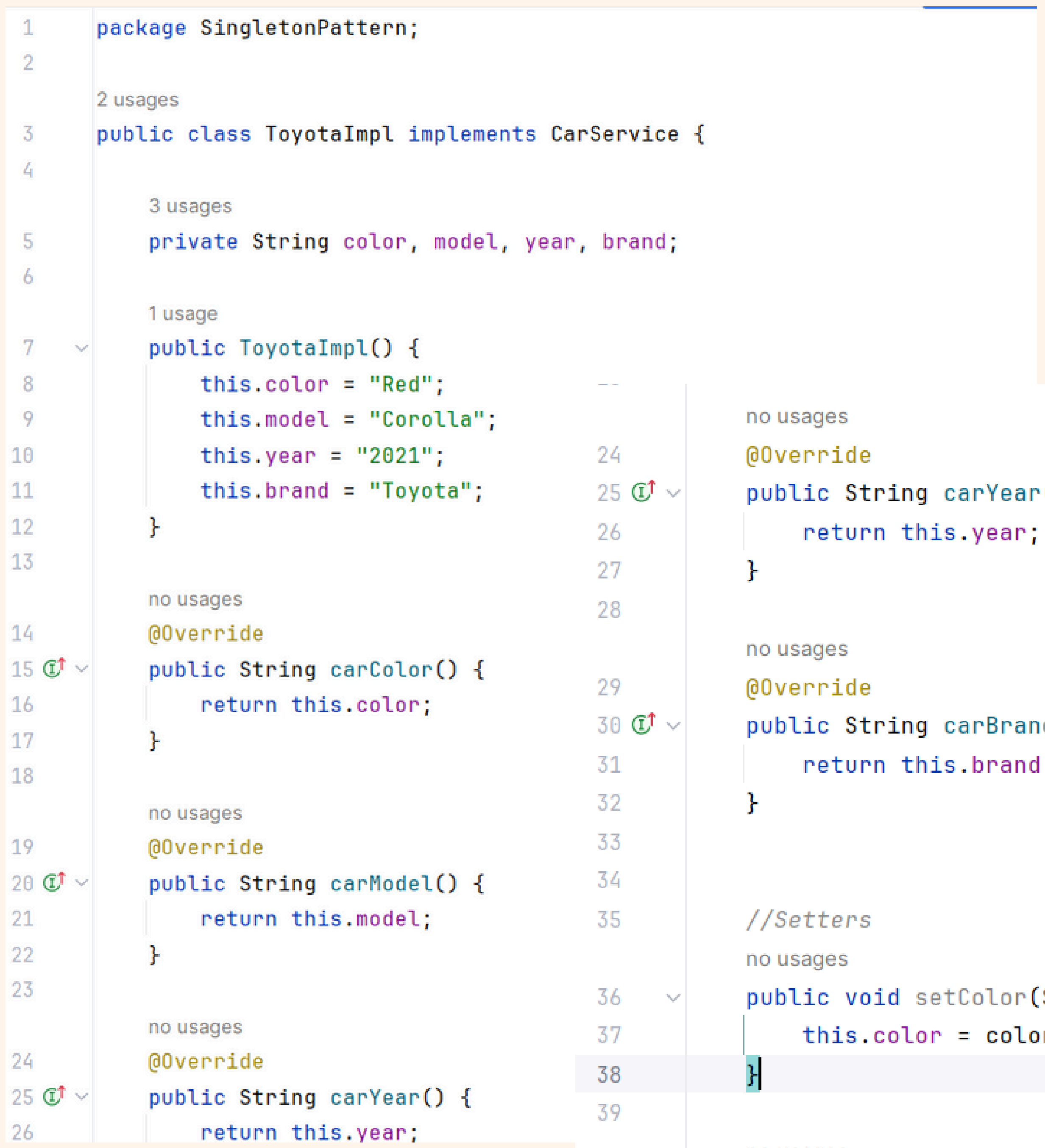


```
package SingletonPattern;

7 usages   1 implementation
public interface CarService {

    no usages   1 implementation
    public String carColor();
    no usages   1 implementation
    public String carModel();
    no usages   1 implementation
    public String carYear();
    no usages   1 implementation
    public String carBrand();
}
```

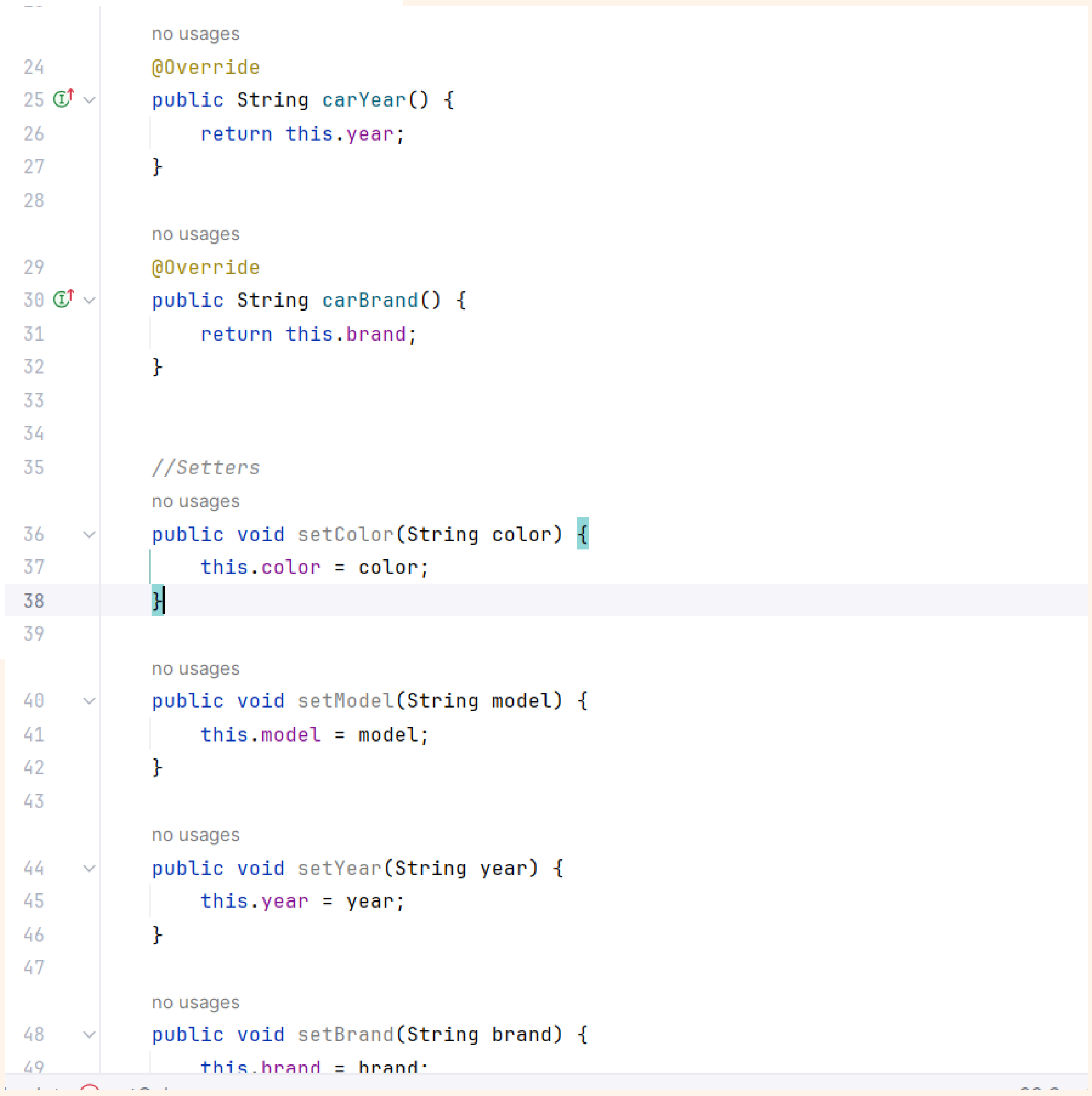
CarService establishes the contract that classes implementations will have to accept.

The beans allocated into our Spring Context will be of type **CarService** because gives it more flexibility in the code when centralizing.

Toyota car class Implementation



```java
package SingletonPattern;

2 usages
public class ToyotaImpl implements CarService {

    3 usages
    private String color, model, year, brand;

    1 usage
    public ToyotaImpl() {
        this.color = "Red";
        this.model = "Corolla";
        this.year = "2021";
        this.brand = "Toyota";
    }

    no usages
    @Override
    public String carColor() {
        return this.color;
    }

    no usages
    @Override
    public String carModel() {
        return this.model;
    }

    no usages
    @Override
    public String carYear() {
        return this.year;
```

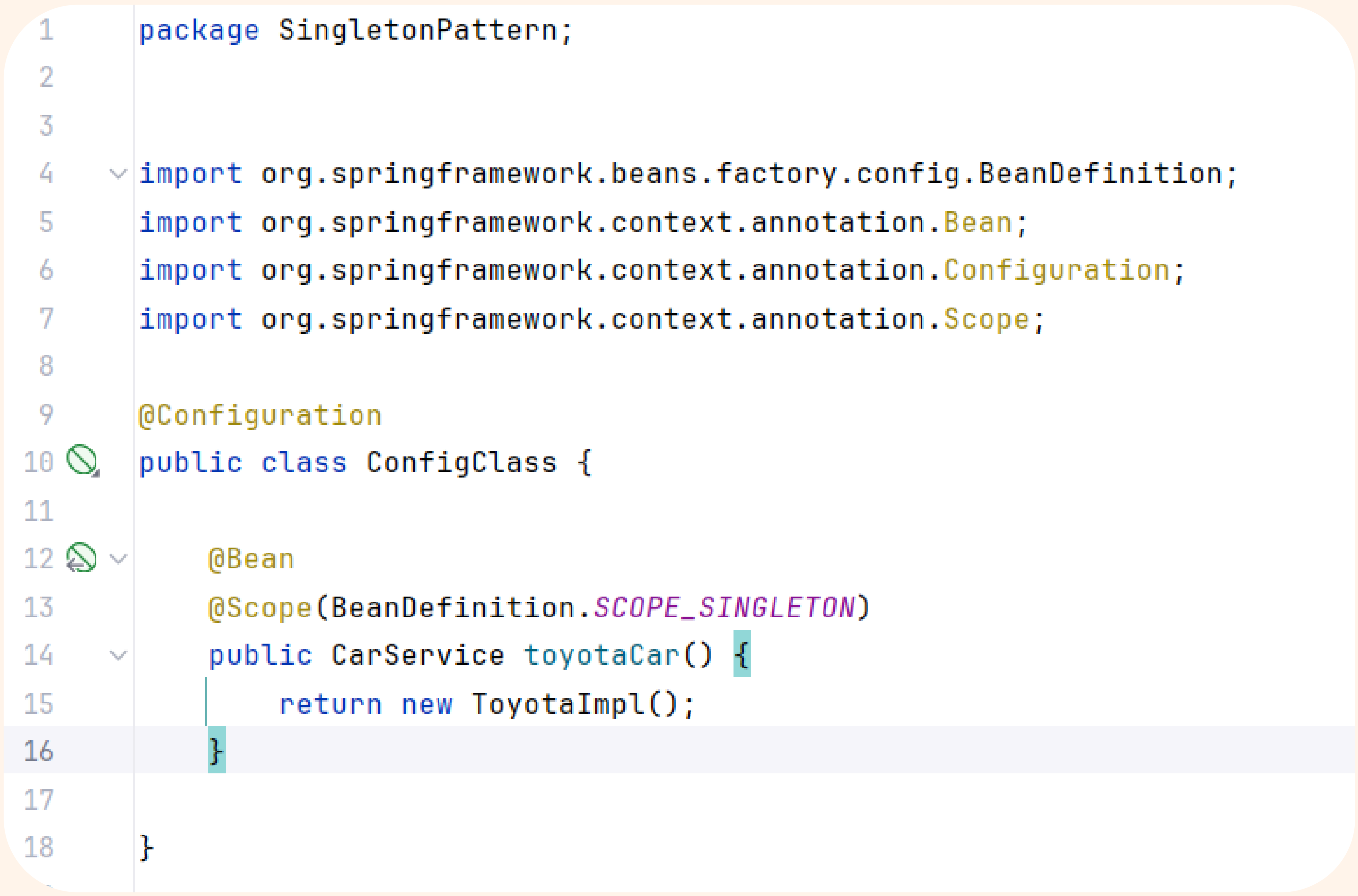


```java
    no usages
    @Override
    public String carYear() {
        return this.year;
    }

    no usages
    @Override
    public String carBrand() {
        return this.brand;
    }


    //Setters
    no usages
    public void setColor(String color) {
        this.color = color;
    }

    no usages
    public void setModel(String model) {
        this.model = model;
    }

    no usages
    public void setYear(String year) {
        this.year = year;
    }

    no usages
    public void setBrand(String brand) {
        this.brand = brand;
```

Configuration Class

08

This is where beans are declared



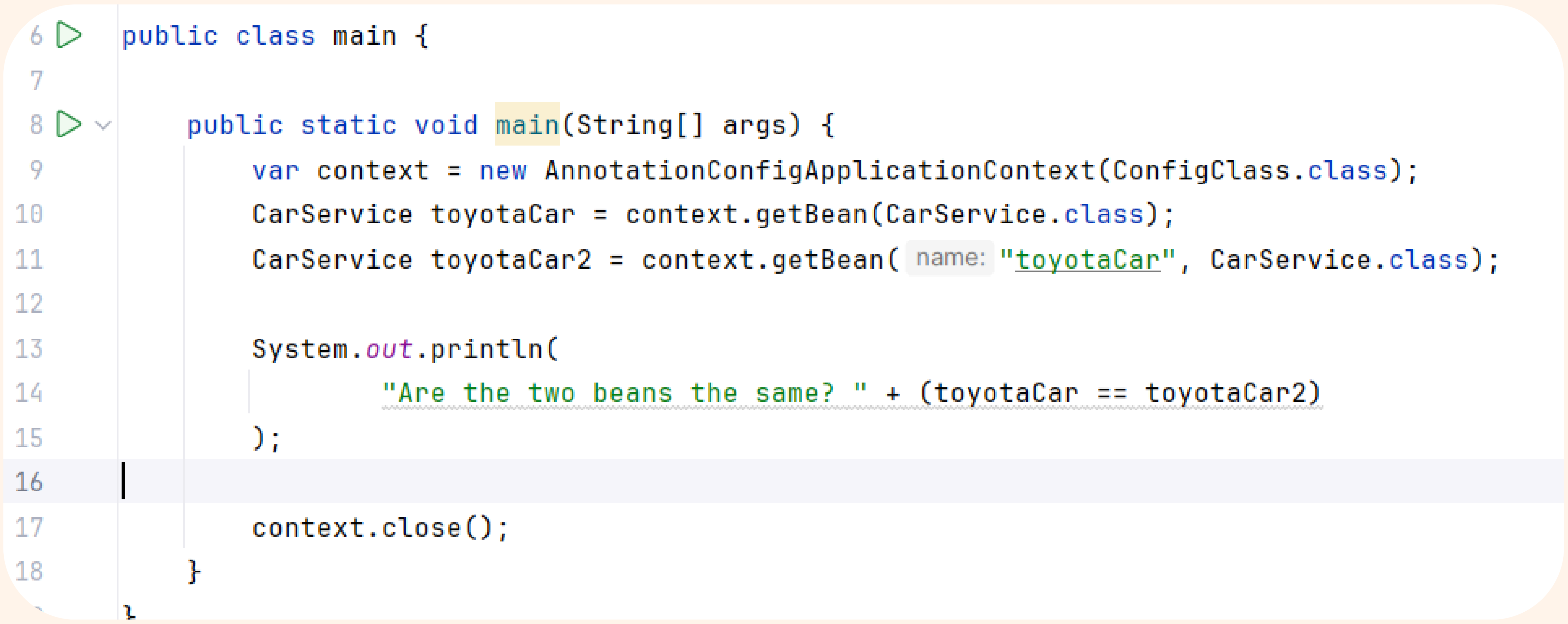
```java
package SingletonPattern;


import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Scope;

@Configuration
public class ConfigClass {

    @Bean
    @Scope(BeanDefinition.SCOPE_SINGLETON)
    public CarService toyotaCar() {
        return new ToyotaImpl();
    }

}
```

Take a look at what we are returning: **the class implementation**. And that will serves us as the instance to be stored in the context.

Main Class

09

We get our instances by invoking the Spring Context.



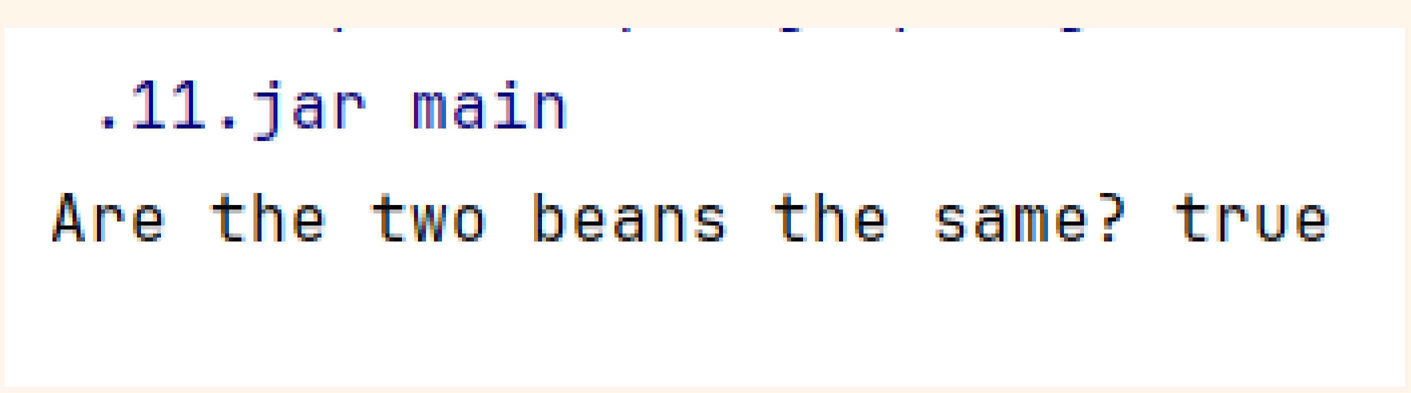
```java
public class main {

    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext(ConfigClass.class);
        CarService toyotaCar = context.getBean(CarService.class);
        CarService toyotaCar2 = context.getBean( name: "toyotaCar", CarService.class);

        System.out.println(
                "Are the two beans the same? " + (toyotaCar == toyotaCar2)
        );

        context.close();
    }
}
```

We are pretending here to compare the two invocations of the same instance to see if both of them are equal or not.

The singleton scope **must meet expectations by throwing a “true” result** in the console; meaning that it treats of the **same instance created** in the container.

Result in Console

10



```
 .11.jar main
Are the two beans the same? true
```

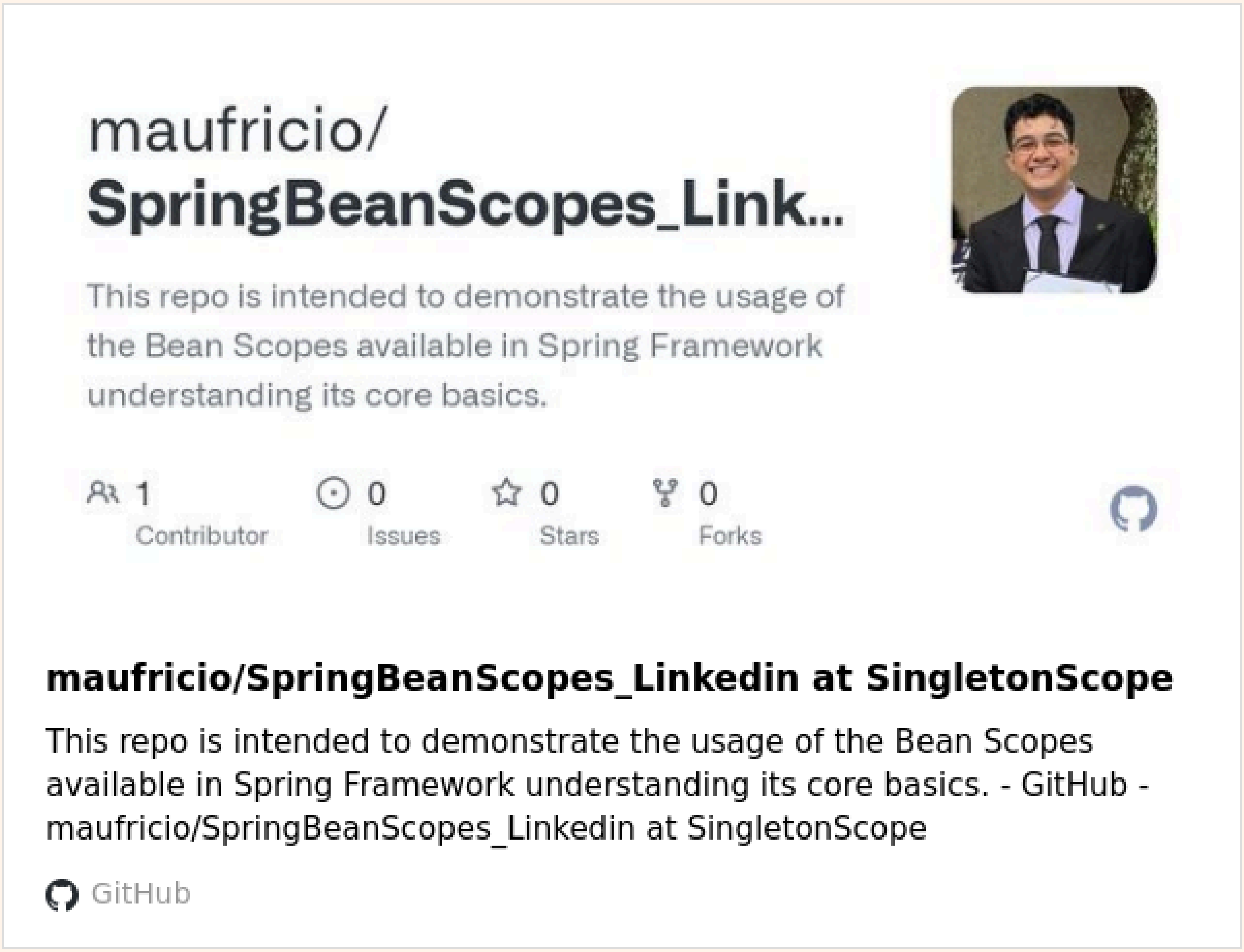
You see?

It is so simple to exemplify but basefuly to understand and to avoid some gaps that might occur when using more advanced topics than this.

Code in my Github Profile

Repo Link:

https://github.com/maufricio/SpringBeanScopes_Linkedin/tree/SingletonScope



maufricio/
SpringBeanScopes_Link...

This repo is intended to demonstrate the usage of the Bean Scopes available in Spring Framework understanding its core basics.

1 Contributor · 0 Issues · 0 Stars · 0 Forks

maufricio/SpringBeanScopes_Linkedin at SingletonScope

This repo is intended to demonstrate the usage of the Bean Scopes available in Spring Framework understanding its core basics. - GitHub - maufricio/SpringBeanScopes_Linkedin at SingletonScope

GitHub

Thank you!

What part of the example do you consider is the most influential?

@mauricioperez

If you liked it, don't hesitate to recommend this post!

Like Celebrate Love Insightful Curious

LinkedIn