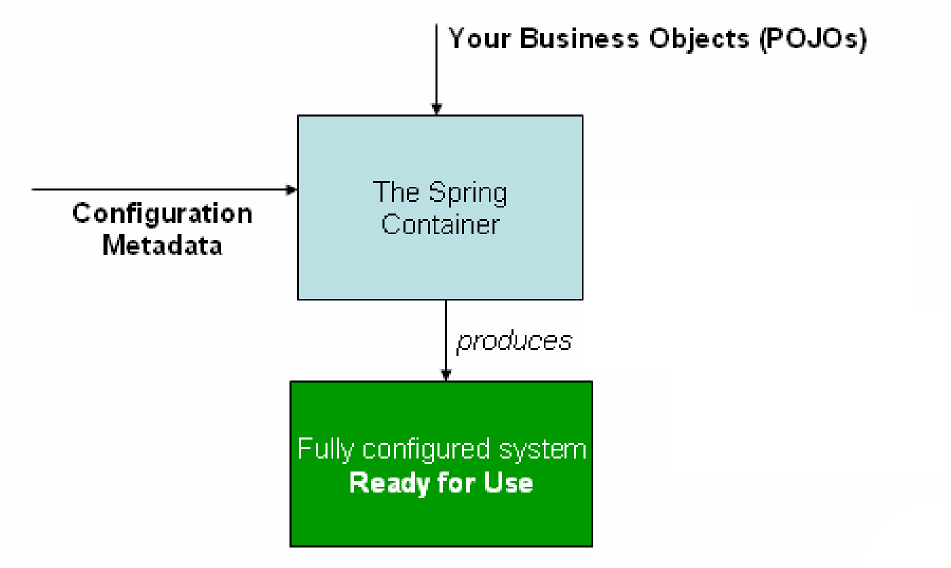
Contents



Spring IOC : Spring IoC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control.

Challenge 1 : What is the difference between Building a project , compiling a project and running a project ?
HINT : https://stackoverflow.com/questions/2650168/building-vs-compiling-java

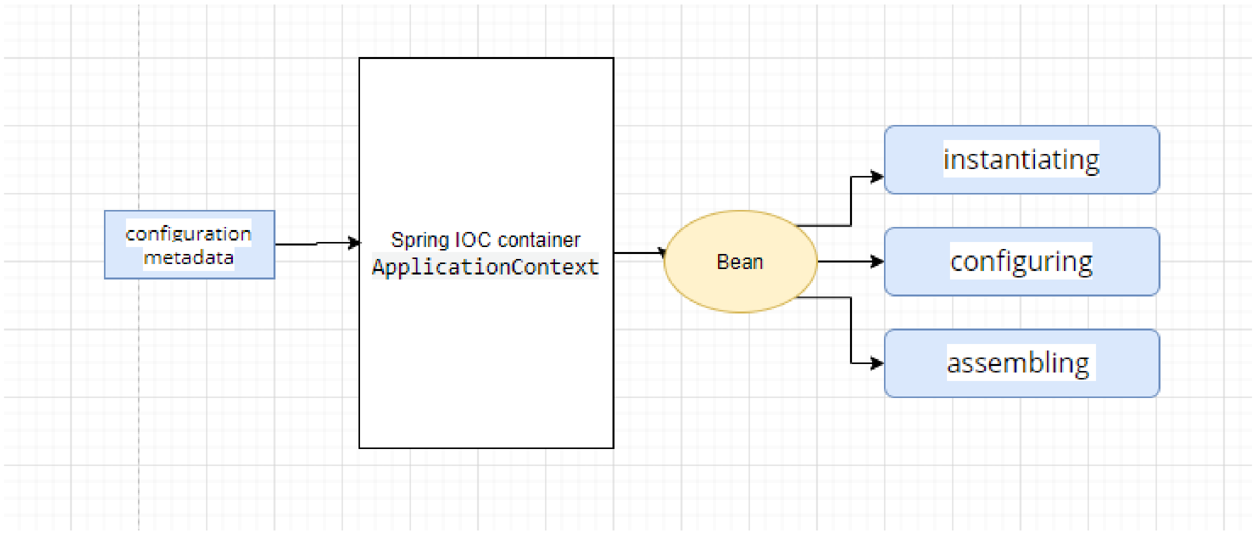
Challenge 2 : What is the significance of @Component annotation ?
HINT : https://www.baeldung.com/spring-component-annotation

Challenge 3 : What are @RestController, @Service , @Repository annotations ?
HINT : These are sophisticated annotations built upon @Component .

Challenge 4 : What Is a Spring Bean?
HINT : In Spring, a bean is anything (NOT NECESSARILY OBJECT as IOC container also has properties) that the Spring container instantiates, assembles, and manages. It has scope . All properties which are beans will have singleton scope .For objects which are beans , scope is singleton by default . For objects , scope can be changed to prototype .
https://stackoverflow.com/questions/17193365/what-in-the-world-are-spring-beans
https://www.baeldung.com/spring-bean-scopes

Challenge 5 : Where is the bean created by the Spring IOC stored ?
HINT :Application Context which is an IOC Container . https://www.baeldung.com/spring-application-context

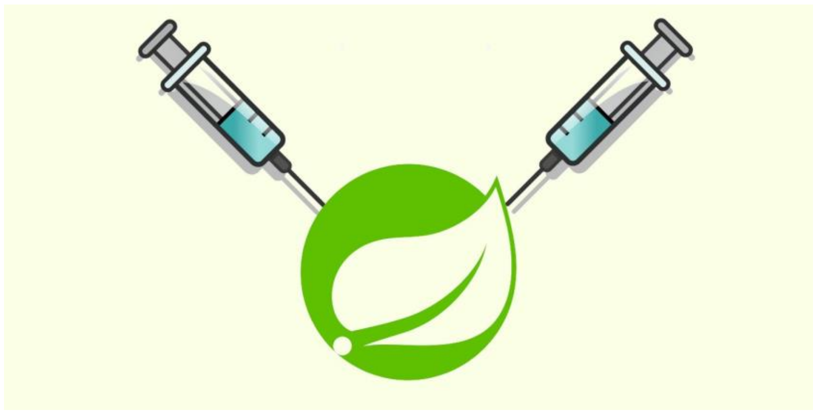
Challenge 6 : Where is the Application Context IOC Container stored , in memory or in the disk of your computer ?
HINT : Think if once your server stops , do your beans stay alive !



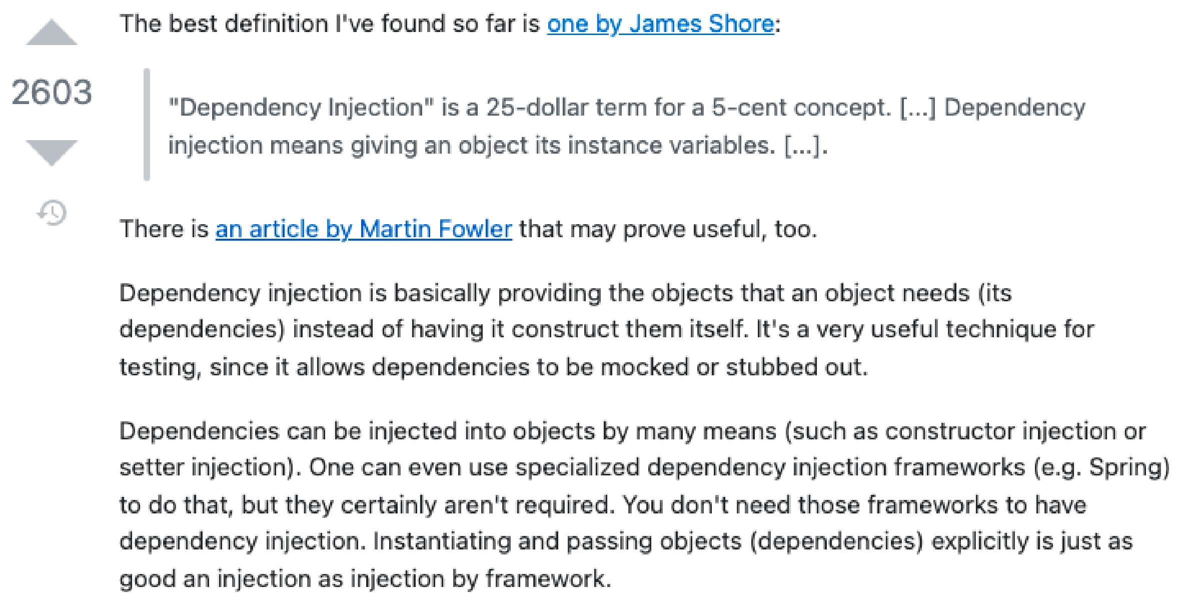
References :

https://www.geeksforgeeks.org/spring-understanding-inversion-of-control-with-example/

Dependency Injection : Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose-coupling between the classes.



Challenge 1 : What is the need for Dependency Injection ?
HINT : Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be dependent on class Two. Now though it might appear okay to depend on one module on the other but, in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided.
Spring IOC resolves such dependencies with Dependency Injection, which makes the code easier to test and reuse. Loose coupling between classes can be possible by defining interfaces for common functionality and the injector will instantiate the objects of required implementation. The task of instantiating objects is done by the container according to the configurations specified by the developer.



2603

The best definition I've found so far is one by James Shore:

> "Dependency Injection" is a 25-dollar term for a 5-cent concept. [...] Dependency injection means giving an object its instance variables. [...].

There is an article by Martin Fowler that may prove useful, too.

Dependency injection is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself. It's a very useful technique for testing, since it allows dependencies to be mocked or stubbed out.

Dependencies can be injected into objects by many means (such as constructor injection or setter injection). One can even use specialized dependency injection frameworks (e.g. Spring) to do that, but they certainly aren't required. You don't need those frameworks to have dependency injection. Instantiating and passing objects (dependencies) explicitly is just as good an injection as injection by framework.

Challenge 2 : There is a class called User and it does not have the Component Annotation directly or indirectly . Will the IOC container have its bean already created which in future can be injected as a dependency to another class ?

Challenge 3 : What is the advantage of Dependency Injection ?
HINT : One single object created as bean in IOC container can be injected in 100 different classes , thus saving space . Also , we have to change the class once and changes reflect in 100 classes .

Challenge 4 : Can DI exist without IOC ?
HINT : If we are unable to create beans , what will we inject? Thus , No .

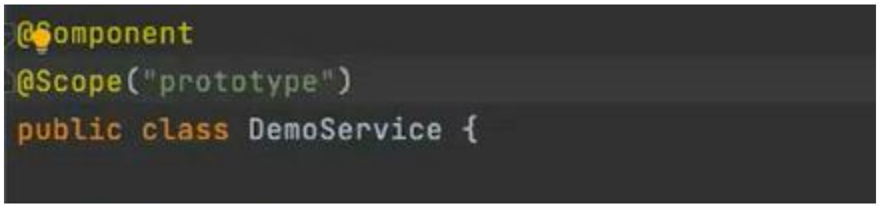
Challenge 5 : Can a bean be injected to a class which is not having Component annotation i.e. it itself is not a bean ?
HINT : No , it will not work . Why - To manage parent dependencies . It's a rule from Spring Developers .

Challenge 6 : If you have a Java project (not maven) , can you add the MySQL dependency through POM ?

Challenge 7 : What is @Autowired Annotation ?
HINT : https://www.geeksforgeeks.org/spring-autowired-annotation/
Remember , dependencies of a class will not get injected if we make an object of that class using the new keyword because Autowiring only works on application startup .

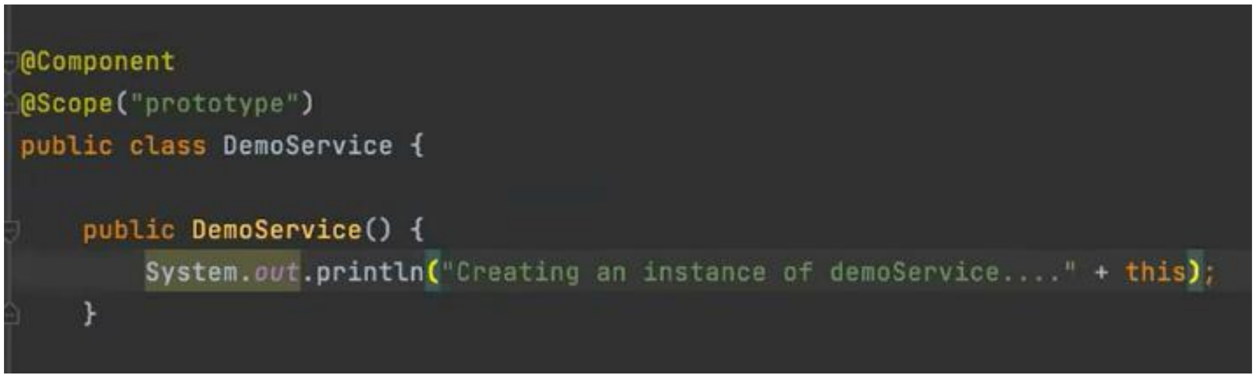
Challenge 8 : I have class A and class B . Both have an autowired dependency of class C . Now tell me is the dependency injected in A and B same or different ?
HINT : You can check the hash codes to verify .

Challenge 9 : (i) How to change scope of your class of how its bean will be created ie as singleton or prototype ?
(ii) How many objects are created at the start of application before changing scope to prototype and before changing scope from singleton ?



```java
@Component
@Scope("prototype")
public class DemoService {
```

HINT : By Default scope is singleton . To change it , you can use @Scope and give prototype as an argument . When scope was singleton , one shared object was created on application startup .



```java
@Component
@Scope("prototype")
public class DemoService {

    public DemoService() {
        System.out.println("Creating an instance of demoService...." + this);
    }
```

On changing the scope to prototype , we can also have a constructor in the class whose scope was changed . In the constructor , we can have a static variable to keep count or print something and then check in the logs and count . But basically wherever your class whose scope is prototype has been autowired, it will create separate objects for each autowiring i.e. dependency injection .

If the scope is prototype , the object of that class will only be created if its referenced (autowired) . If no other class has your prototype scoped class as dependency , no object will be created on the start of application .

Challenge 10 : Suppose if I have a class A with @Component annotation . Class B extends A . Does class B inherit the @Component annotation from A ?

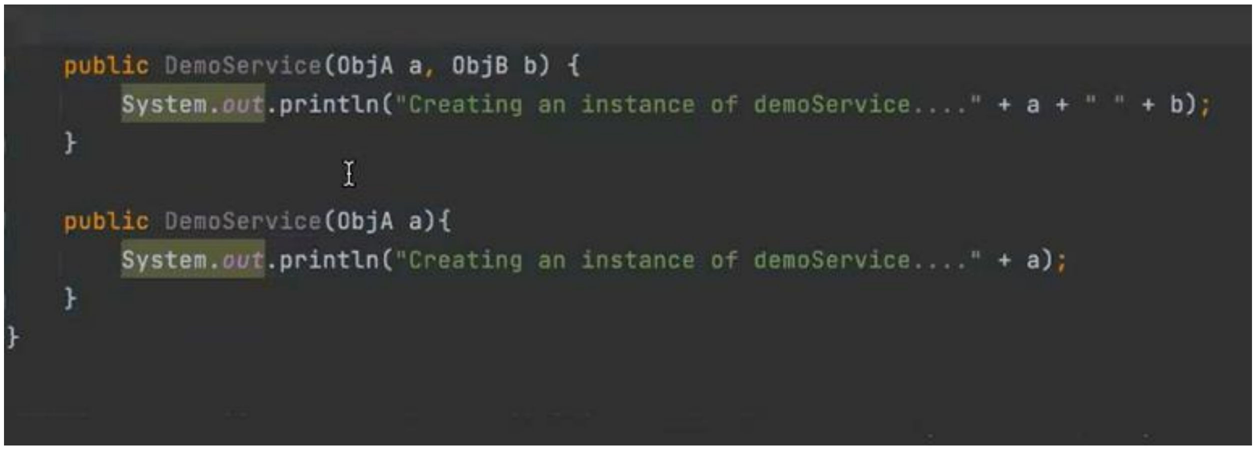
Challenge 11 : Does the dependency of the prototype scope class stay inside the IOC container ? HINT : No .

Challenge 12 (Homework) : Suppose Parent class has a dependency of Class A injected . Now the Child class inherits from the Parent class . Both have @Component annotation . Will the injected dependency be inherited by the Child class in (i)A has singleton scope (ii)A has prototype scope

Challenge 13 : If dependency of class A is being autowired into class B and class C . You can have any scope . What happens if I define a parameterised constructor in class A and run the application ?
HINT : It will fail as bean of class A is created by Spring and it does not know which parameters to provide . If in case you need to define a parameterised constructor , then whenever you are defining a parameterized constructor of a bean, you always need to pass bean(s) in the constructor arguments . Because Spring is cognizant of them .

Challenge 14 : (Homework) If we have both default constructor and parameterised constructor defined with 2 possible cases : (i) primitive argos (ii) beans as args , what will happen in each case ?

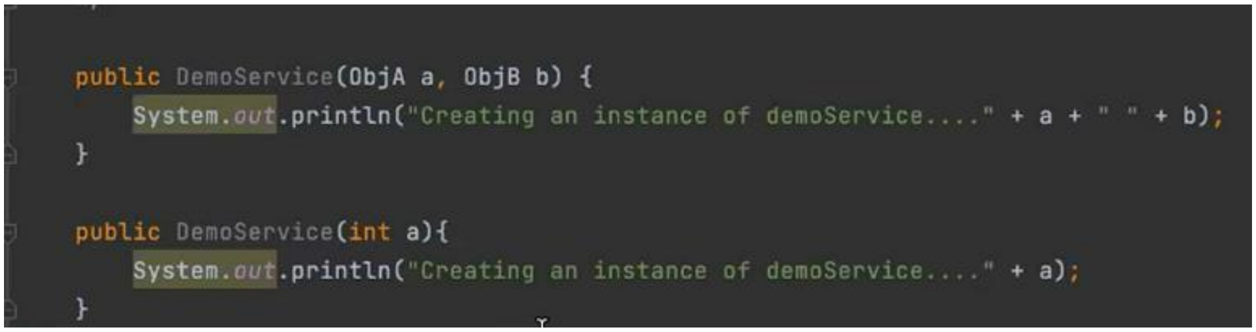
Challenge 15 : (Homework) Now we know that if we have a parameterised constructor and we give beans as arguments , it works fine . But what if we have more than one parameterised constructor as in the below example ?



```
    public DemoService(ObjA a, ObjB b) {
        System.out.println("Creating an instance of demoService...." + a + " " + b);
    }

    public DemoService(ObjA a){
        System.out.println("Creating an instance of demoService...." + a);
    }
}
```

HINT : It will fail because again Spring has no clue which parameterised constructor to call .

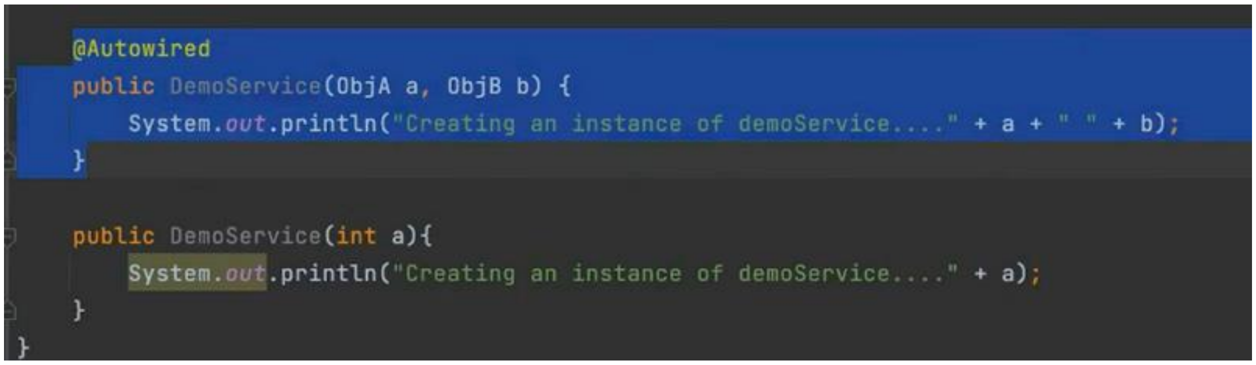
Challenge 16 : (Homework) Will this case work having 2 parameters constructors , one with beans as args and other as primitives args ? As Spring can recognize constructor with beans as args , should this work ?



```java
    public DemoService(ObjA a, ObjB b) {
        System.out.println("Creating an instance of demoService...." + a + " " + b);
    }

    public DemoService(int a){
        System.out.println("Creating an instance of demoService...." + a);
    }
```

HINT : Spring will not see that if out of 2 parameterised constructor , one has beans which it could have recognised . It sees more than one parameterised constructor and throws errors as Spring can not deal with ambiguity .

Challenge 17 : How to solve the above problem without deleting any constructor and not defining the default constructor ? HINT : You can tell spring which param constructor to use .



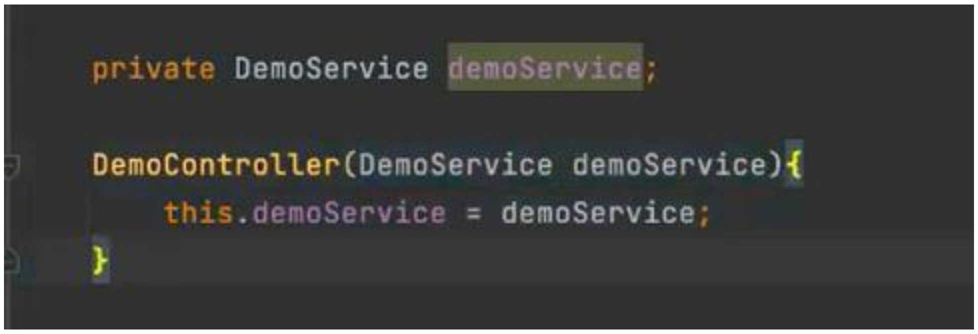
```java
    @Autowired
    public DemoService(ObjA a, ObjB b) {
        System.out.println("Creating an instance of demoService...." + a + " " + b);
    }

    public DemoService(int a){
        System.out.println("Creating an instance of demoService...." + a);
    }
}
```

Challenge 18 : Can we add Autowired annotation to more than one constructor ?
HINT : No

Challenge 19 : What are the types of Dependency Injection ?
HINT : 1. Constructor Injection https://www.baeldung.com/constructor-injection-in-spring
2. Setter Injection https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

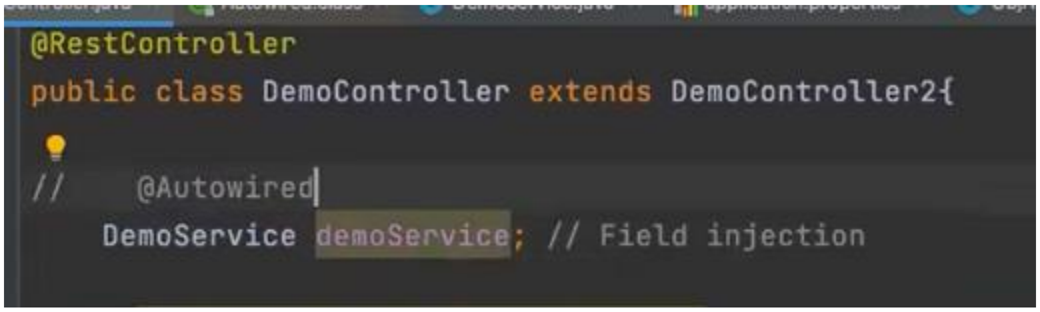
Challenge 20 : Can we achieve Constructor Injection only through the Autowired annotation ?

HINT : No , The below configuration will work .In case of one parameterized constructor , it will work as Autowired annotation is just a marker annotation and tells Spring which constructor to use , which is obvious to Spring already in this case .



```
    private DemoService demoService;

    DemoController(DemoService demoService){
        this.demoService = demoService;
    }
```

Challenge 21 : If we remove the Autowired in the below field injection , will it work . Did the dependency of DemoService get injected in DemoController ?

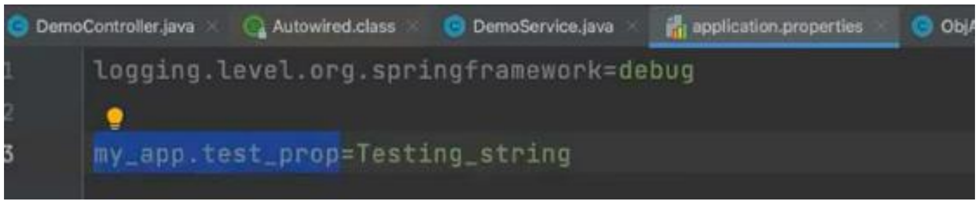


```
@RestController
public class DemoController extends DemoController2{

//    @Autowired
    DemoService demoService; // Field injection
```

HINT : No , Here Autowired annotation is not just working as marker annotation .Autowired basically tells Spring to get the DemoService bean from IOC container .

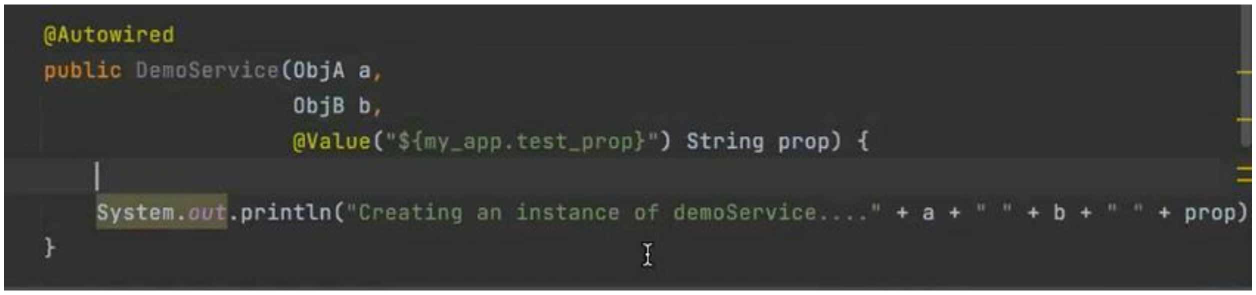
Challenge 22 : How to make Spring familiar with your primitive variable in Parameterised constructor ?

HINT : Step 1 - By defining it in our application.properties provided by Spring . The property becomes a bean .



DemoController.java | Autowired.class | DemoService.java | application.properties | ObjA

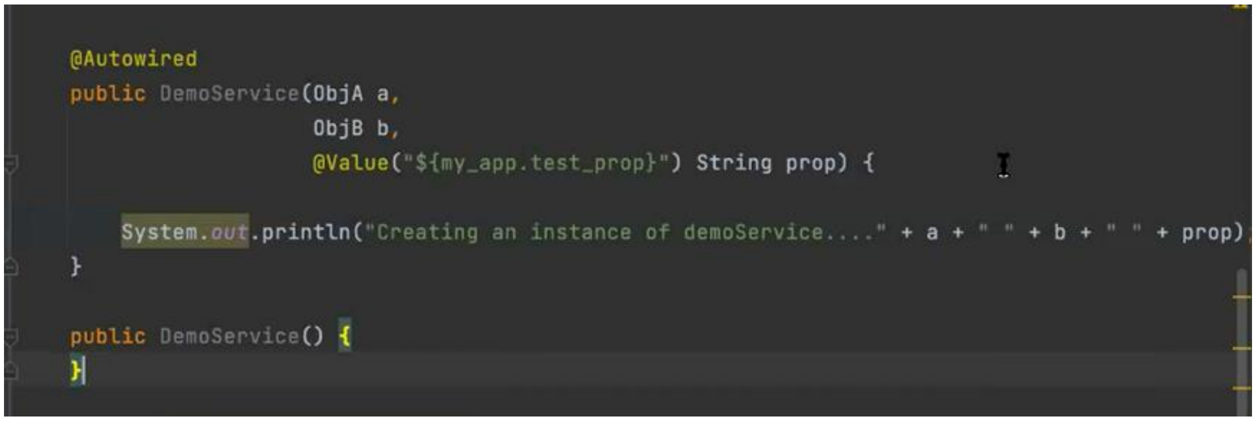
```
logging.level.org.springframework=debug

my_app.test_prop=Testing_string
```



```
@Autowired
public DemoService(ObjA a,
                   ObjB b,
                   @Value("${my_app.test_prop}") String prop) {

    System.out.println("Creating an instance of demoService...." + a + " " + b + " " + prop)
}
```

Step 2 - And then in our class for which we are creating a constructor , we can use @Value annotation to use the value defined in application.properties . Note that this property is now a bean . Also we have Autowired our constructor below only because we had more than one constructor .This signals to Spring to make sure it uses this one specifically .If we have only one constructor , Autowired annotation is not required .

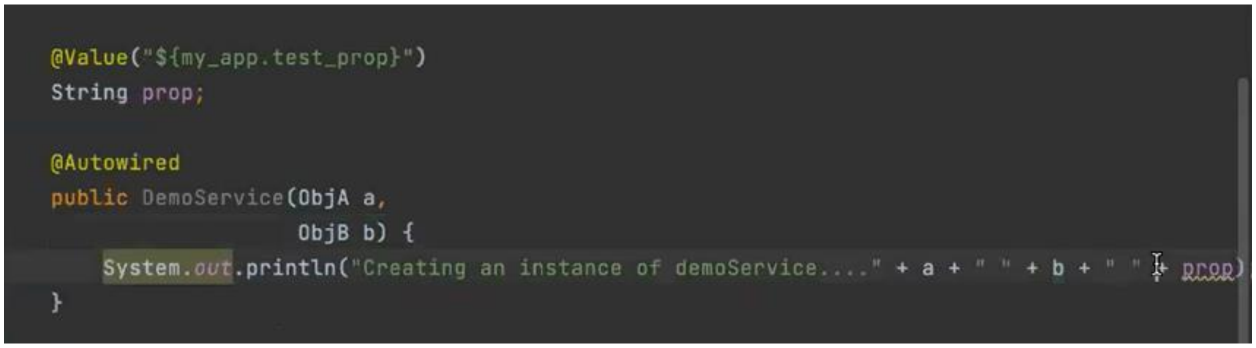
Challenge 23 : (i)In this case which constructor of DemoService is called when its bean is made ? (ii) In the second case , which constructor is called if we remove the Autowired annotation ?



```
@Autowired
public DemoService(ObjA a,
                   ObjB b,
                   @Value("${my_app.test_prop}") String prop) {

    System.out.println("Creating an instance of demoService...." + a + " " + b + " " + prop);
}

public DemoService() {
}
```

HINT : Case 1 - parameterised , Case 2 - default

Challenge 24 : In the below DemoService class , we defined our property from application.property on the top using @Value . Now the question is when the constructor of DemoService is called , what will be the value printed for prop ?

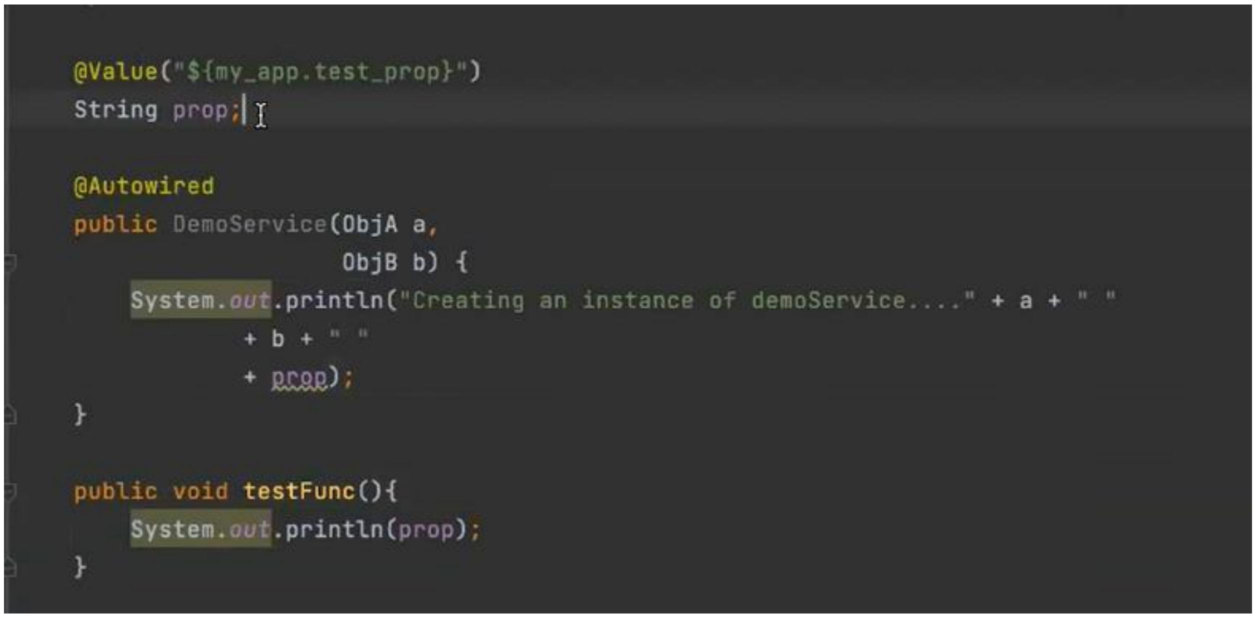


```
@Value("${my_app.test_prop}")
String prop;

@Autowired
public DemoService(ObjA a,
                   ObjB b) {
    System.out.println("Creating an instance of demoService...." + a + " " + b + " " + prop)
}
```

HINT : null . Because in Java , first the constructor are created , then only properties are initialized . Example :

```
class GFG {

 private Integer num = 4;

    public GFG() {
            System.out.println("Creating an instance of demoService....”+num); // null
          }

 }
```

Challenge 25 : Will the value of prop be null , if we call it inside a function too ?



```
@Value("${my_app.test_prop}")
String prop;

@Autowired
public DemoService(ObjA a,
                   ObjB b) {
    System.out.println("Creating an instance of demoService...." + a + " "
            + b + " "
            + prop);
}

public void testFunc(){
    System.out.println(prop);
}
```

HINT : No

References :

https://www.geeksforgeeks.org/spring-dependency-injection-with-example/

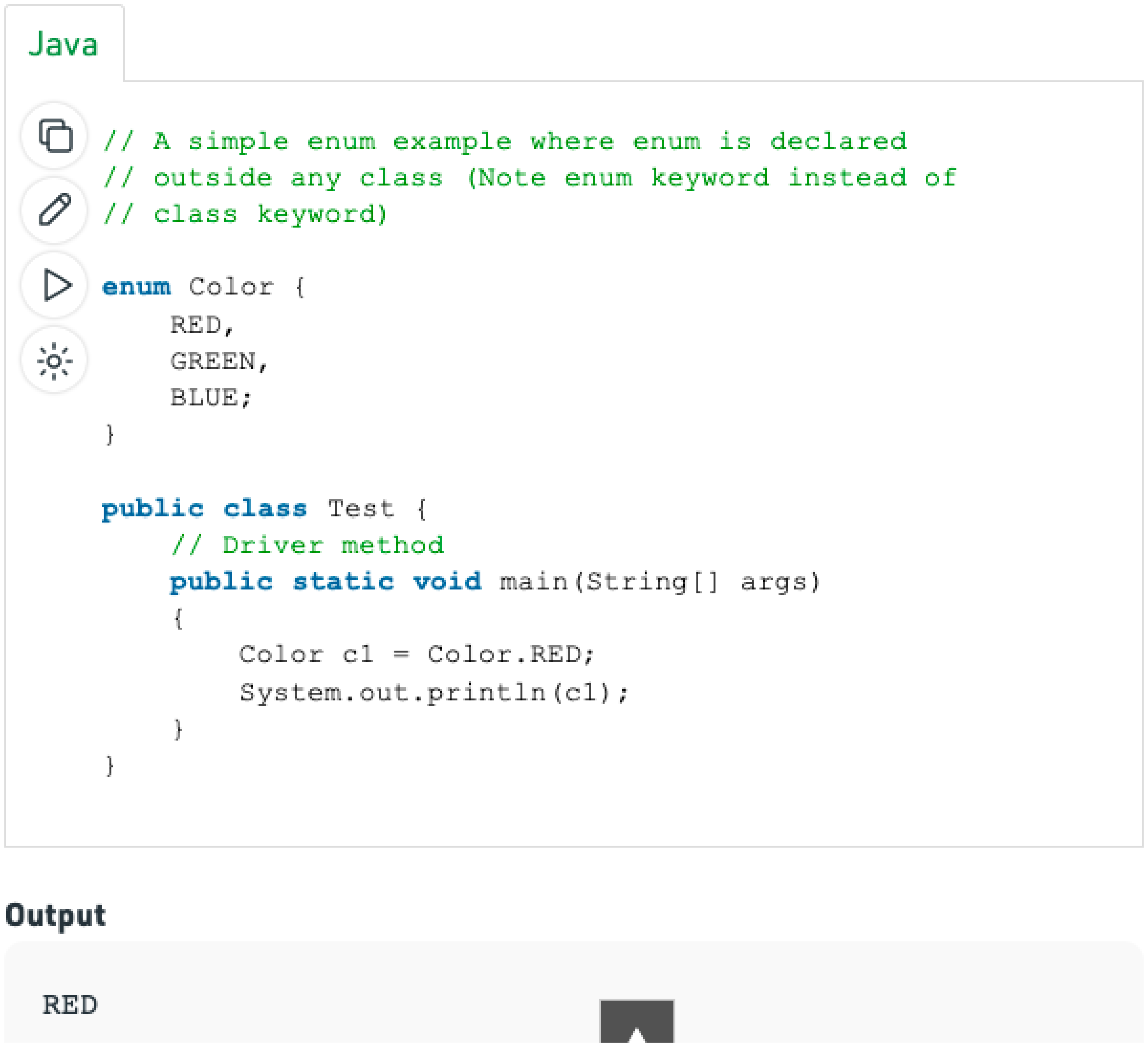
https://martinfowler.com/articles/injection.html

https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

https://www.baeldung.com/spring-autowire

https://www.geeksforgeeks.org/spring-value-annotation-with-example/

ENUMS : Enumerations serve the purpose of representing a group of named constants in a programming language. For example, the 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit. Other examples include natural enumerated types (like the planets, days of the week, colors, directions, etc.). Enums are used when we know all possible values at compile-time, such as choices on a menu, rounding modes, command-line flags, etc. It is not necessary that the set of constants in an enum type stay fixed for all time. https://www.geeksforgeeks.org/enum-in-java/



```java
// A simple enum example where enum is declared
// outside any class (Note enum keyword instead of
// class keyword)

enum Color {
    RED,
    GREEN,
    BLUE;
}

public class Test {
    // Driver method
    public static void main(String[] args)
    {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

Output

```
RED
```

TARGET IN ANNOTATIONS

Let's switch to JDK 1.8 as it has better documentation than JDK 11 . And then lets read the documentation to understand the target . If you use annotation at wrong target , you get a compile time error .

Challenge 1 : What is the correct target for @RequestParam ?

Challenge 2 : Can we autowire a class ? HINT : No , check target of @Autowired annotation .

Challenge 3 : What is the meaning of Target - ElementType.Type ?

CONFIGURATION OF BEANS WITH @Configuration and @Bean

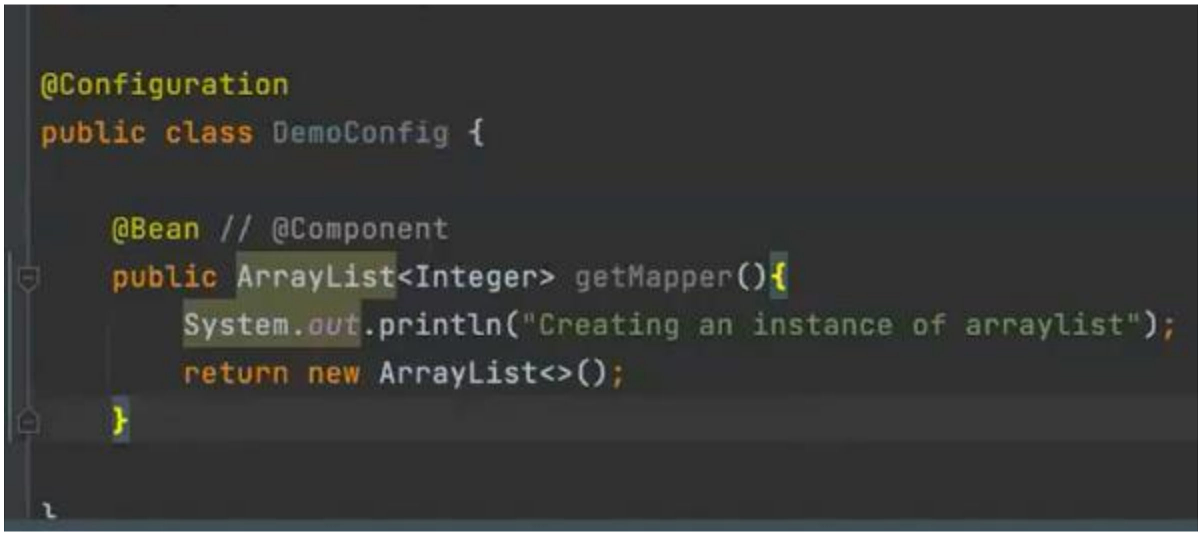
Suppose I have a class in source code for which I don't have write access . But I need to make a bean of that class and store it in the IOC container . How can we do this ?
We can do this through creating a configuration class having a class level annotation of @Configuration . Inside it , I can have my methods returning beans .Each of those methods will have a method level @Bean annotation .

https://howtodoinjava.com/spring-core/spring-configuration-annotation/
https://zetcode.com/spring/bean/

https://www.geeksforgeeks.org/spring-bean-annotation-with-example/

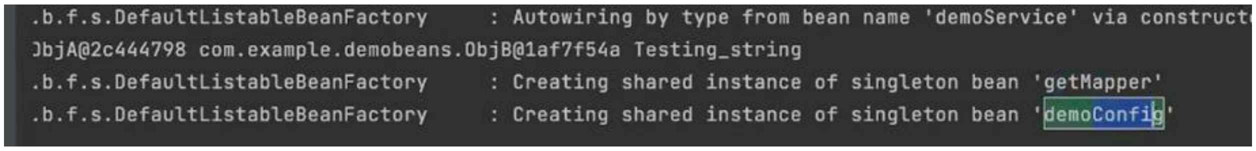
Challenge 1 : What will be the scope of bean created using @Bean ?
HINT : singleton

Challenge 2 : Can we create a bean of ArrayList class or can we make an arraylist as a bean ?
HINT :



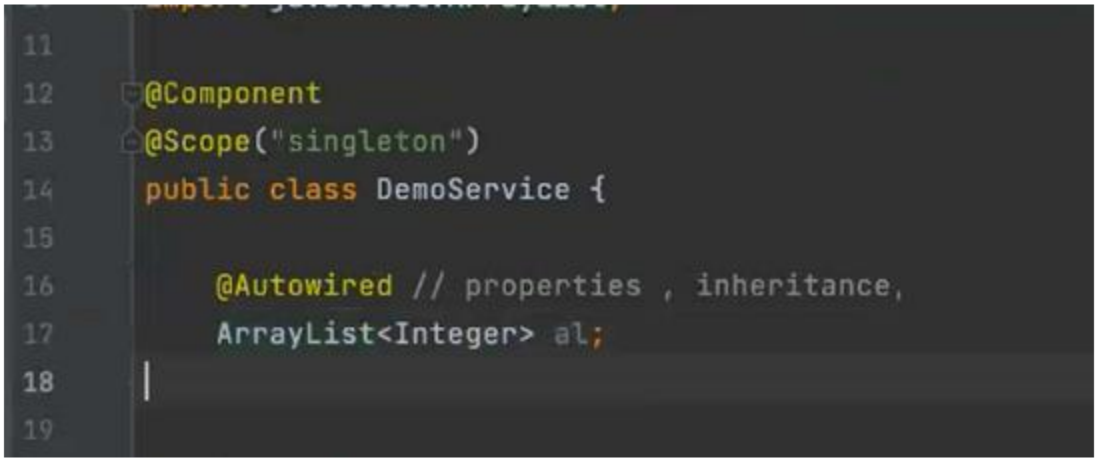
```java
@Configuration
public class DemoConfig {

    @Bean // @Component
    public ArrayList<Integer> getMapper(){
        System.out.println("Creating an instance of arraylist");
        return new ArrayList<>();
    }

}
```

On running , we can see in the logs …



```
.b.f.s.DefaultListableBeanFactory     : Autowiring by type from bean name 'demoService' via constructo
ObjA@2c444798 com.example.demobeans.ObjB@1af7f54a Testing_string
.b.f.s.DefaultListableBeanFactory     : Creating shared instance of singleton bean 'getMapper'
.b.f.s.DefaultListableBeanFactory     : Creating shared instance of singleton bean 'demoConfig'
```

Now you can autowire this arraylist in whichever class you need its dependency.



```java

@Component
@Scope("singleton")
public class DemoService {

    @Autowired // properties , inheritance,
    ArrayList<Integer> al;


```

Challenge 3 : (HomeWork) Now suppose we have a configuration class DemoConfig with a bean method called getMapper returning the bean of ArrayList class . Earlier we were autowiring the ArrayList in our 2 controller class DemoController1 and DemoController2 . What if we don't autowire arrayList but rather autowire the DemoConfig in our DemoController1 and DemoController2 and then call the demoConfig.getMapper().add(10) function using the injected object of DemoConfig and then we print the arrayList . All of this logic happens on hitting the GetMapping("/") in both controller classes .What will be the contents of the arrayList in each case if the / api is hit 5 times in both cases :
(i)If the scope of getMapper method is singleton
(ii) If the scope of getMapper method is prototype

HINT : When scope is singleton , even when ArrayList is not autowired but as DemoConfig is autowired so all beans inside @Configuration are put inside IOC container on app startup .ArrayList bean is singleton.

When scope is prototype , no bean is put in IOC on startup . When api is hit , it creates a new object of ArrayList for each api call . Bean has prototype scope and is new every time.

Challenge 4 : (HomeWork) Suppose in the above example , if we have scope of DemoConfig as prototype and scope of getMapper bean method as singleton . We have autowired DemoConfig in our controllers DemoController1 and DemoController2 . What will happen when the app will startup ?

HINT : Rule is that if we have a method level scope defined , then class level scope does not matter . There will be only one object of ArrayList created on startup which will be a singleton bean .There will be two different instances of DemoConfig injected into our two controllers but both instances will refer to the same arrayList bean .
Note : You will not get any logs for creation of DemoConfig instances as logs are only generated for singleton shared instances .

Challenge 5 : What if we have multiple bean methods with the same return type suppose ArrayList<Integer> . Now in our controllers , if we autowire ArrayList<Integer> . What will happen when we start the application ?
HINT : error . So we can use @Primary annotation on top of the bean method we want to specifically inject in case there are other bean methods with the same return type .

References :

https://howtodoinjava.com/spring-core/spring-configuration-annotation/

https://zetcode.com/spring/bean/

https://www.geeksforgeeks.org/spring-bean-annotation-with-example/

https://www.geeksforgeeks.org/bean-life-cycle-in-java-spring/

https://www.geeksforgeeks.org/singleton-and-prototype-bean-scopes-in-java-spring/

https://www.geeksforgeeks.org/how-to-create-a-spring-bean-in-3-different-ways/

https://www.geeksforgeeks.org/spring-inheriting-bean/

https://www.geeksforgeeks.org/difference-between-spring-mvc-and-spring-boot/

https://www.geeksforgeeks.org/difference-between-spring-boot-starter-web-and-spring-boot-starter-tomcat/

LIVE
Java
Backend
Development