

**Lab Manual- Dependency Injection using Spring Boot ( Application Context and @Component)**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab003

**Contributor:**

Contents

[1. Objective 3](#_Toc171619002)

[1. . Traditional Object Creation in Java 3](#_Toc171619003)

[2. Object Creation in Spring Boot 4](#_Toc171619004)

[2. Creating a Spring Boot Project 4](#_Toc171619005)

[3. Import Spring Boot Project in IntelliJ 6](#_Toc171619006)

[4. Understand Default Core SpringBoot Annotation 9](#_Toc171619007)

[5. Create simple package in IntelliJ 11](#_Toc171619008)

[6. Create simple myConfig class in IntelliJ 12](#_Toc171619009)

[7. Create Service Class ProductService in IntelliJ 14](#_Toc171619010)

[8. Create Bean for Class productService under config class in IntelliJ 16](#_Toc171619011)

[9. Call Bean Class inside Main Application 16](#_Toc171619012)

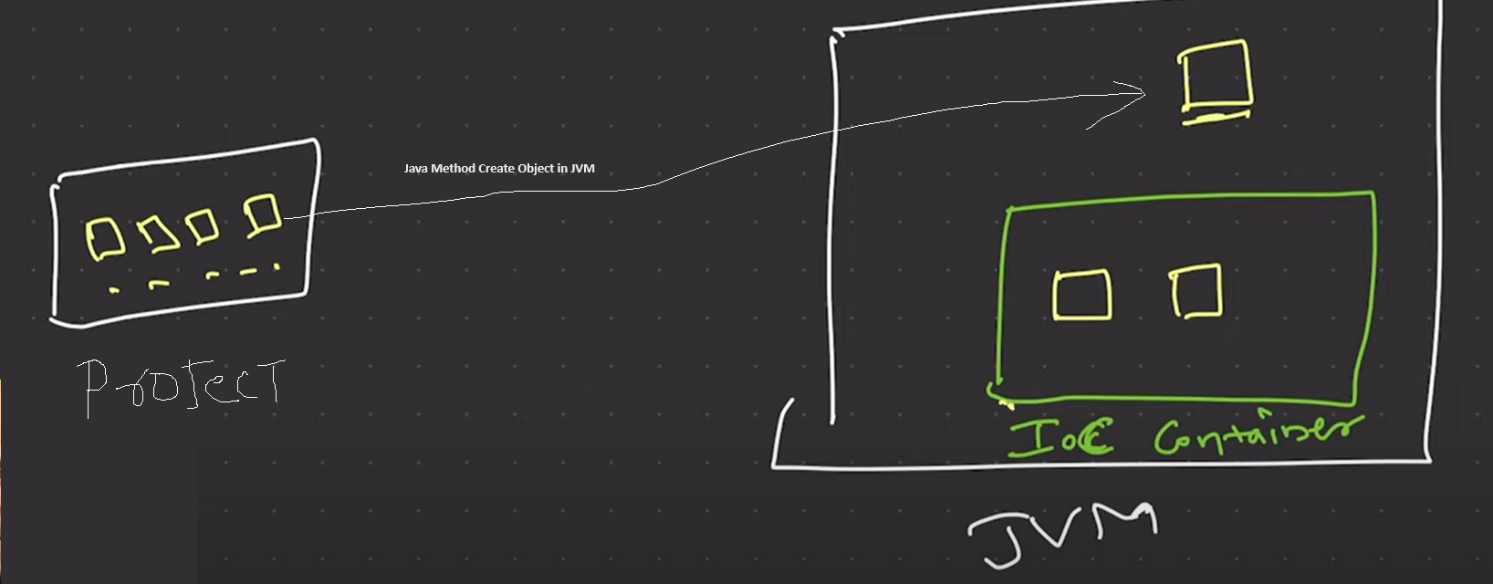
# Objective

In Java, you typically create objects using the new keyword, which directly creates an instance of a class in the JVM. For example:

**MyClass myObject = new MyClass();**

This creates a new instance of MyClass and assigns it to the variable myObject.

In contrast, Spring Boot (and the Spring Framework in general) uses a dependency injection container (often referred to as the Spring container) to manage the lifecycle and dependencies of beans (objects).



Here are some key points:

### . Traditional Object Creation in Java

When you use the new keyword in a traditional Java application:

1. **Direct Creation**: You are directly responsible for creating and managing the lifecycle of the object.
2. **Dependencies**: You have to manually handle dependencies between objects.
3. **Tight Coupling**: The code is often tightly coupled, making it harder to manage and test.

Example:

java

Copy code

public class UserService {

private UserRepository userRepository;

public UserService() {

this.userRepository = new UserRepository();

}

}

### Object Creation in Spring Boot

In Spring Boot, objects (beans) are typically created and managed by the Spring container. This provides several benefits:

1. **Dependency Injection**: Spring manages dependencies between beans, injecting them where needed.
2. **Loose Coupling**: Spring encourages loose coupling between components, making the codebase more modular and easier to test.
3. **Lifecycle Management**: Spring manages the lifecycle of beans, including their creation, initialization, and destruction.

To create a bean in Spring Boot, you typically use annotations like @Component, @Service, @Repository, or @Controller, and let Spring handle the creation and wiring.

Example:

java

Copy code

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

private final UserRepository userRepository;

@Autowired

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

// Business logic methods

}

In this example, UserService is annotated with @Service, indicating that it is a Spring-managed bean. The UserRepository dependency is injected by Spring through the constructor, thanks to the @Autowired annotation. The Spring container will manage the lifecycle of these beans, ensuring they are properly created, initialized, and destroyed.

# Creating a Spring Boot Project

**Step 1:** Open the Spring initializr [https://start.spring.io](https://start.spring.io/).

**Step 2:** Provide the **Group** and **Artifact** name. We have provided Group name **com.bank** and Artifact **as below**

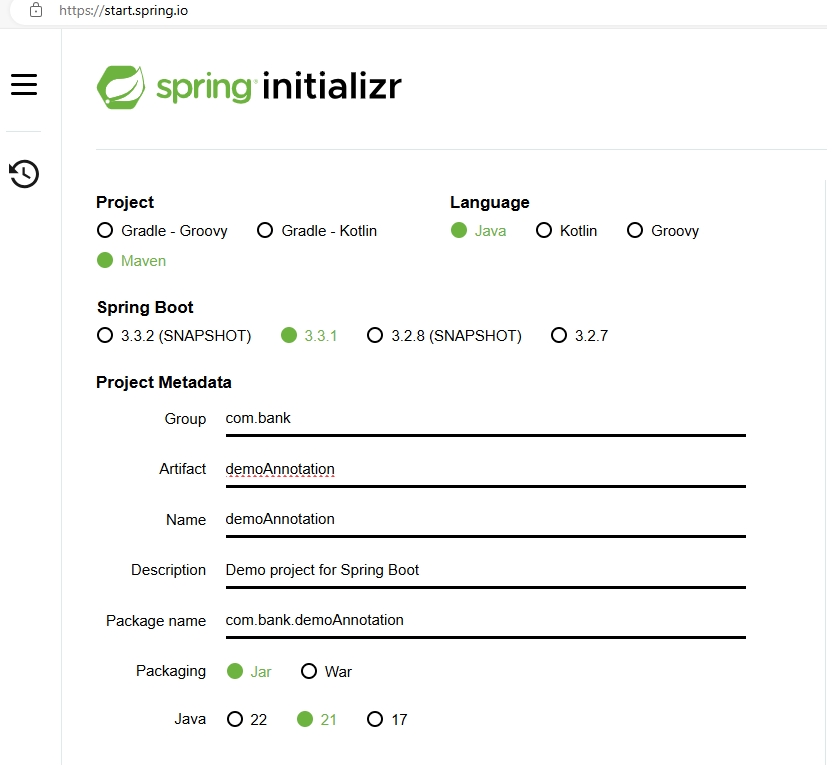
**Group** : com.bank

**Artifact** : demoAnnotation

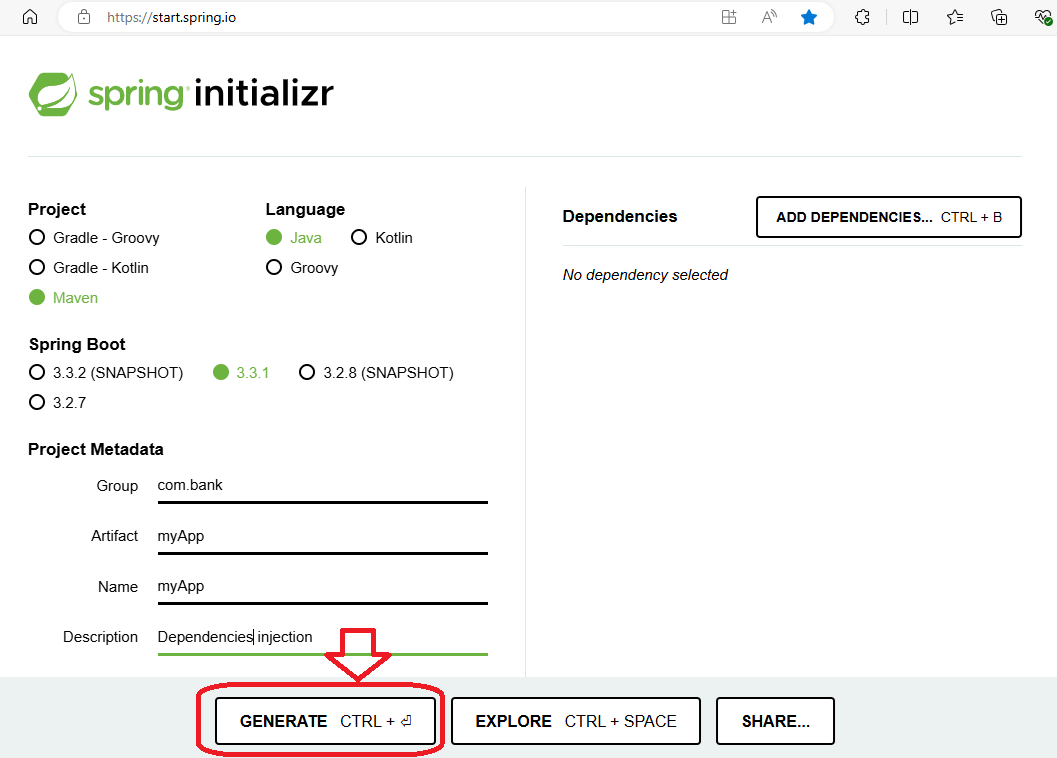
**Name** : demoAnnotation

**Description** : Dependencies Injection

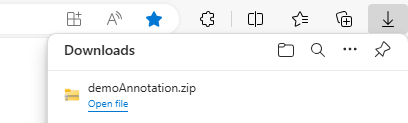
**PackageName** : com.bank. demoAnnotation



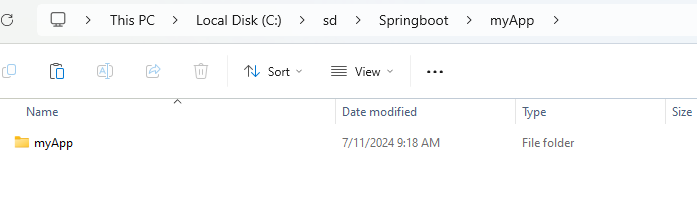
**Step 3: Click on GENERATE to download the code**



**Step 4:** Code should download now

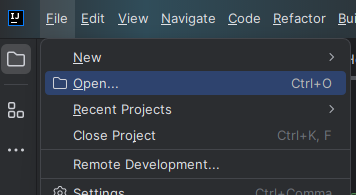


**Step 5: Extract** the project to your Project directory

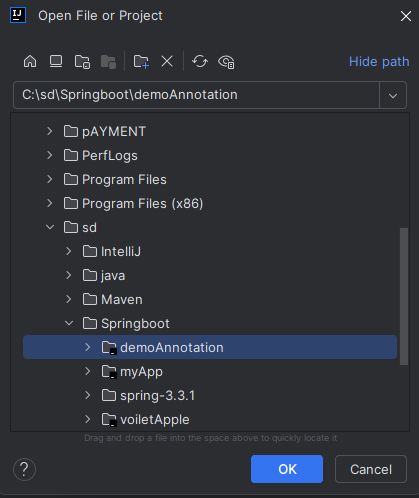


# Import Spring Boot Project in IntelliJ

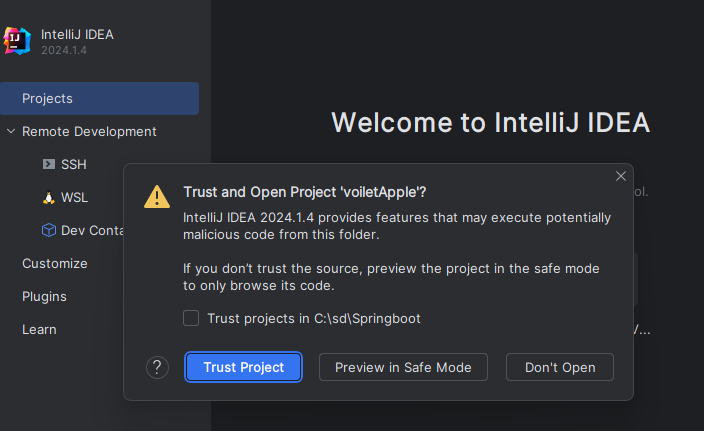
* Click File🡪 Open project

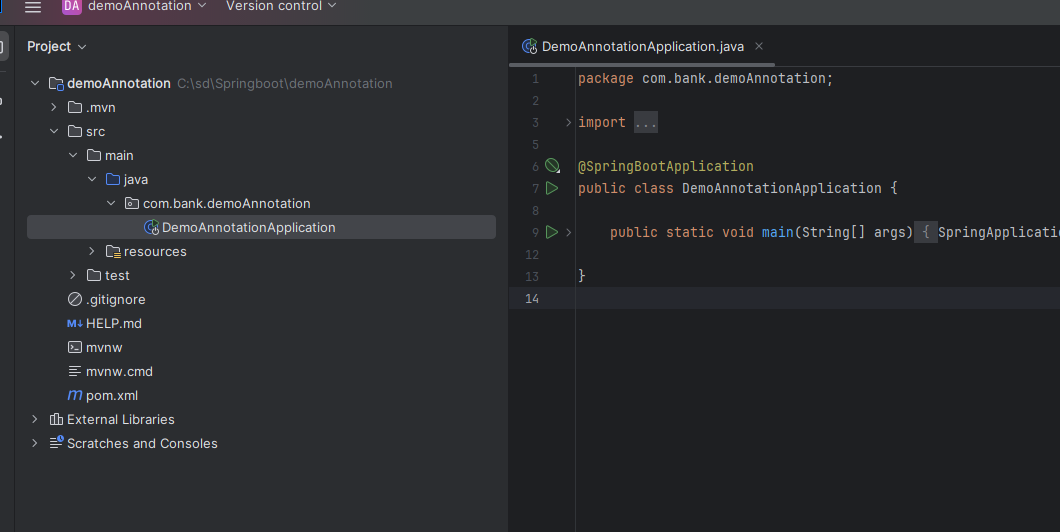


* Select your extract project and click **OK**

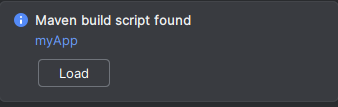


* If ask click **Trust project**

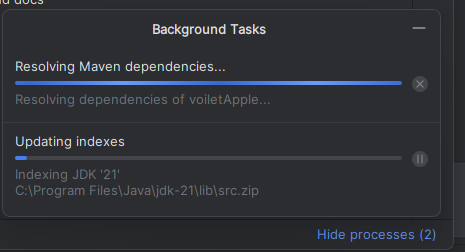




* If ask **Load** the Maven Build Script

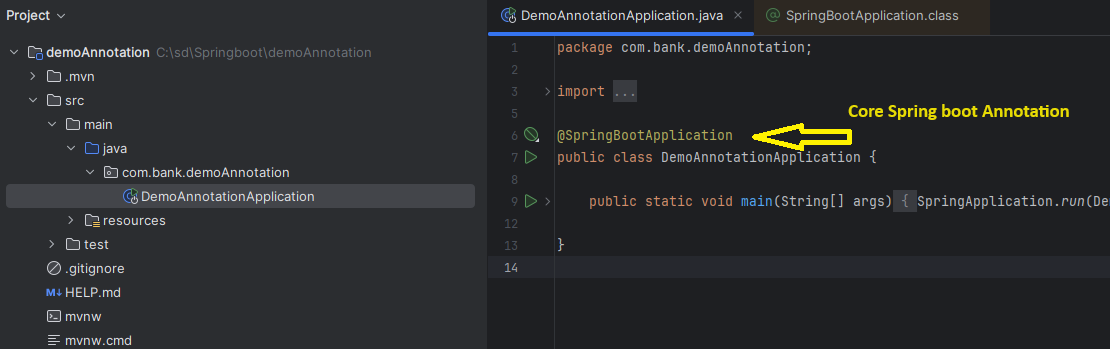


* It may takes couple of Minutes to download and Load the library

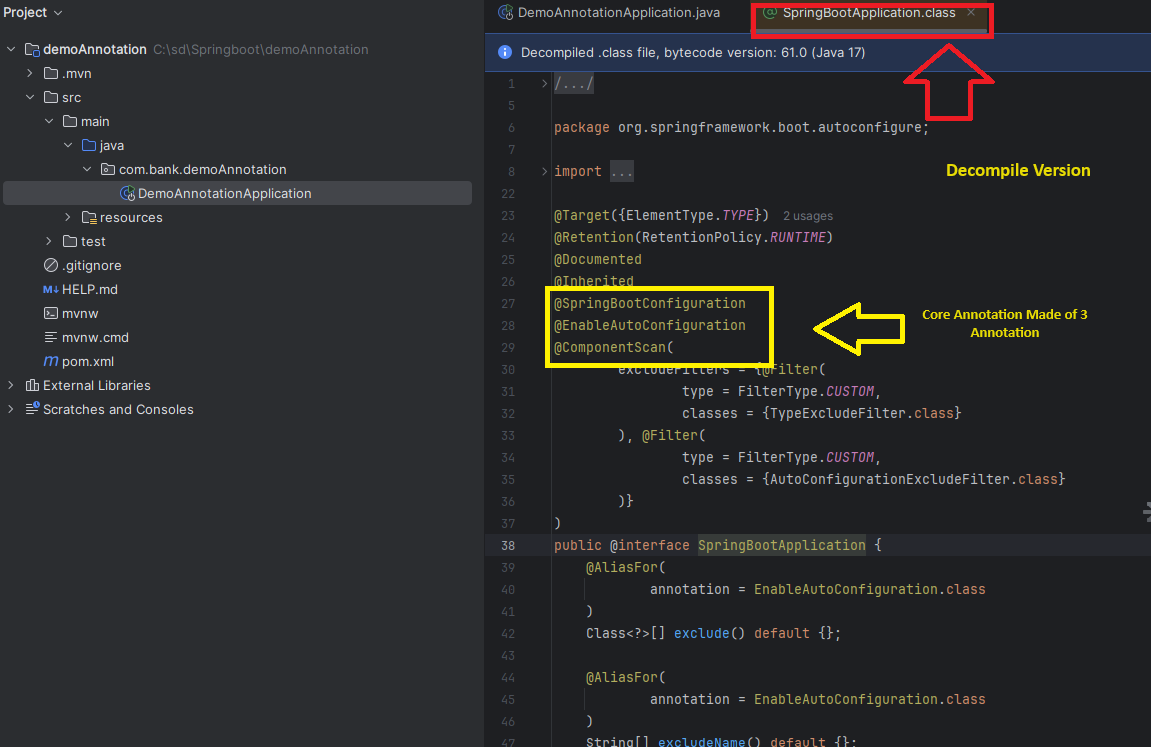


# Understand Default Core SpringBoot Annotation

Press CTRL + Click on @SpringBootApplication



It opens Decompile version of **@SpringBootApplication**

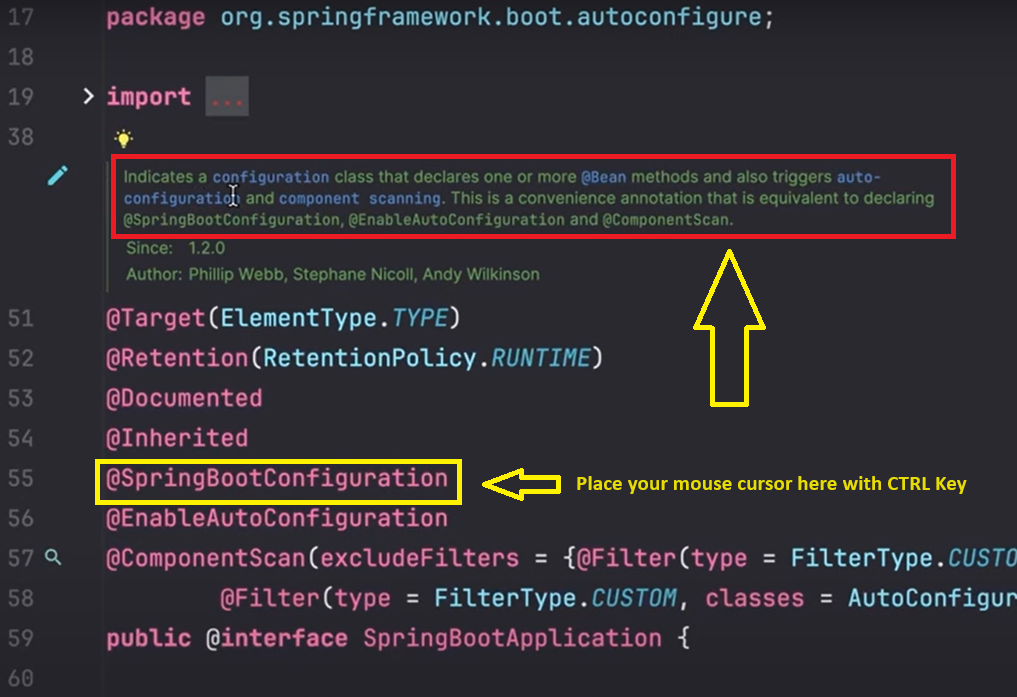
****

When you take the Mouse cursor to **@SpringBootConfiguration** with Press of **+CTRL** button from keyboard **it display the definition**

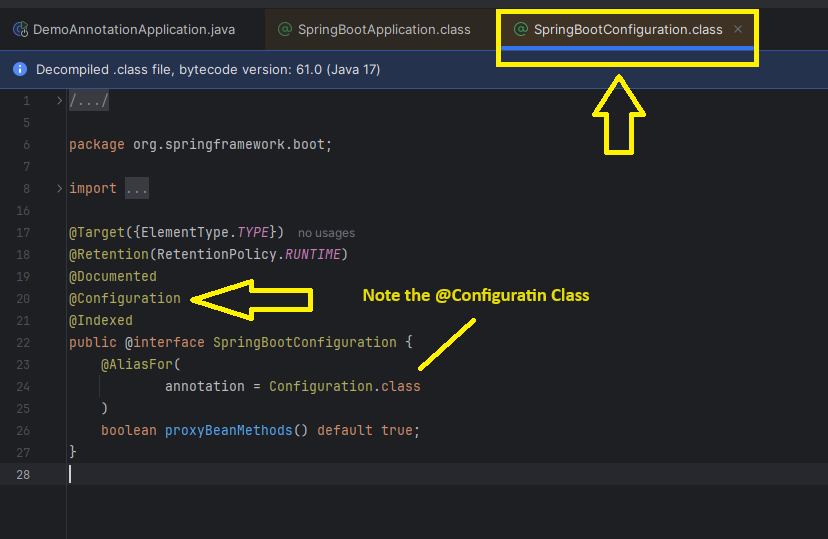
**@Configuration:** It is a class-level annotation. The class annotated with **@Configuration** used by Spring Containers as a source of bean definitions.

**@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

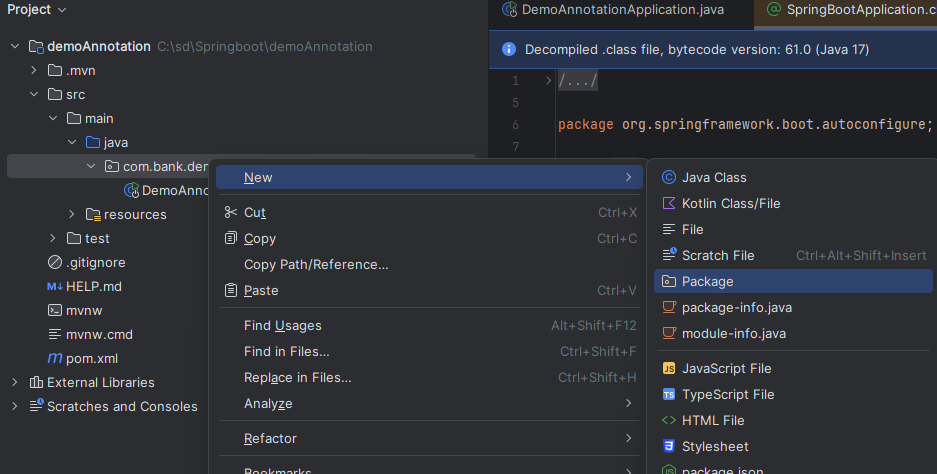
****

You Should Notice **@SpringBootConfiguration** and Clicking it will take you to Next screen where you can note **@configuration** class

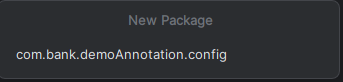


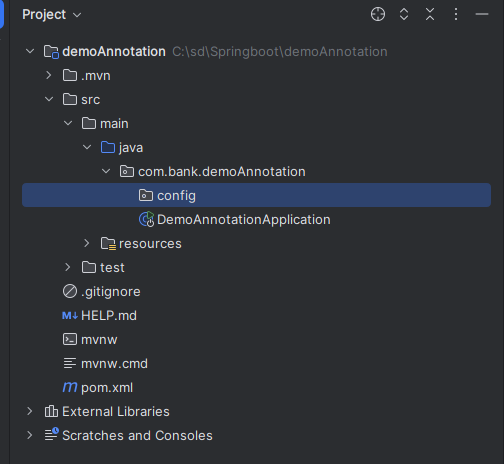
# Create simple package in IntelliJ

Right click the main package com.dev.demoAnnotation 🡪 New 🡪 Package



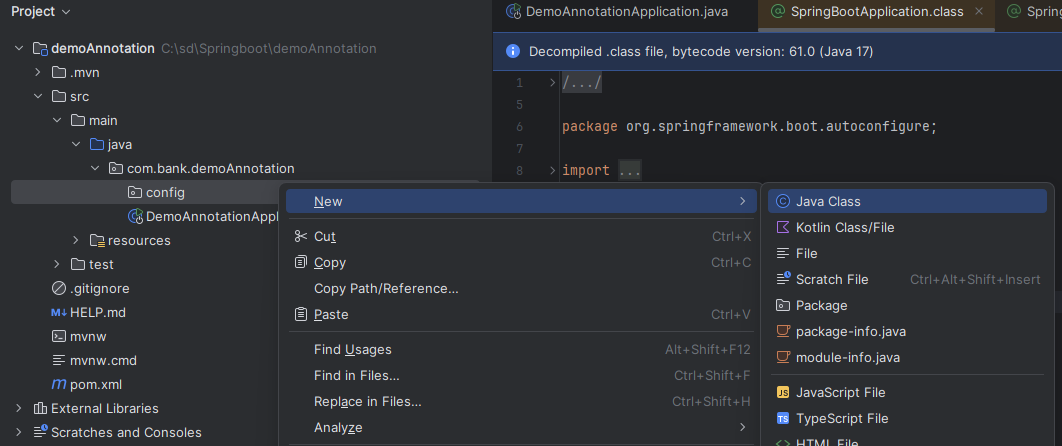
Give the package name **config** and Press Enter



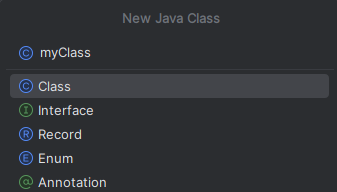


# Create simple myConfig class in IntelliJ

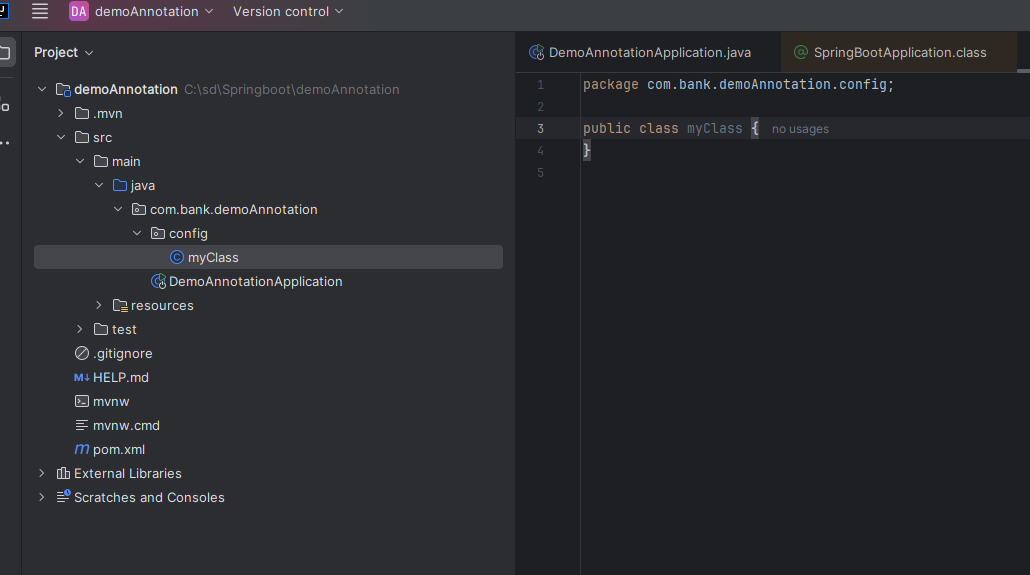
* Right Click the **Config** Package –New—Java Class



* Type **myClass** in Class Name and Press **enter** to create the class

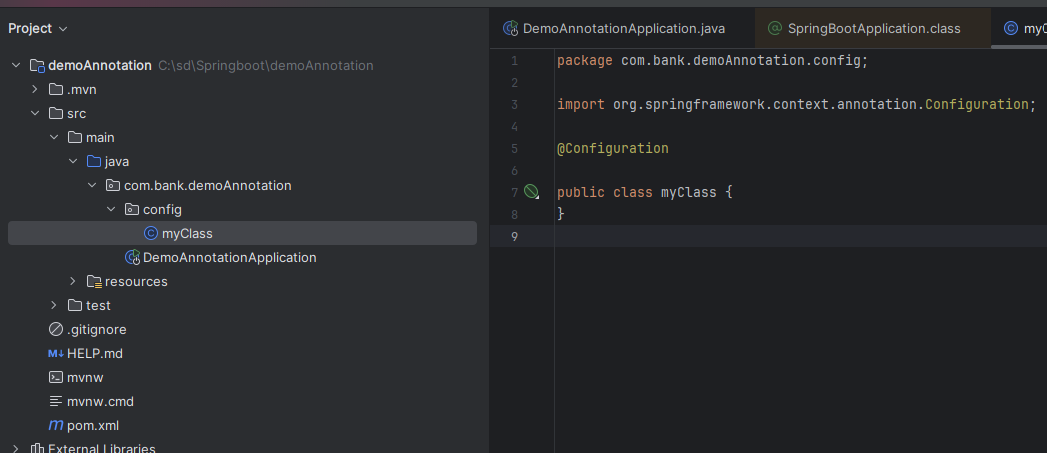


* The myConfig **class** created .
* Note that this is **normal class** not the **config class** , so even if we create **Beans** inside , spring scanner can not search it.



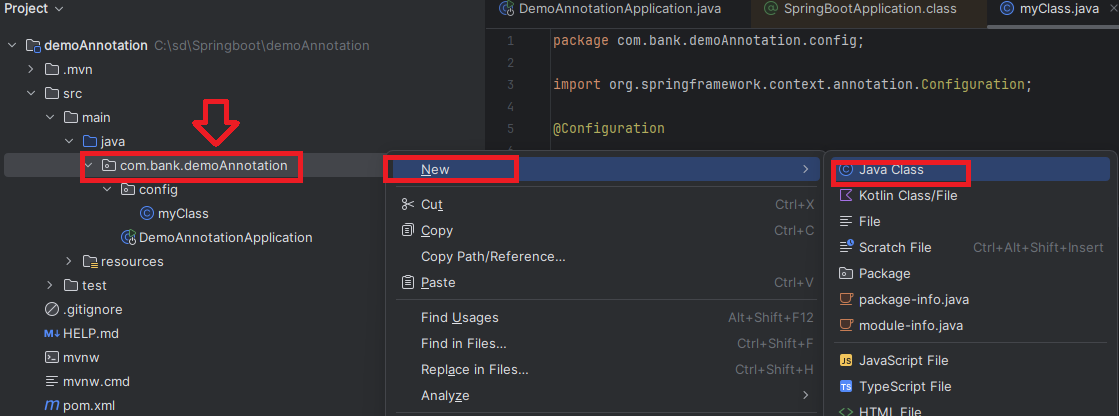
* So to make it as configuration class we have to use annotation **@configuration**

package com.bank.demoAnnotation.config;  
  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
  
public class myClass {  
}

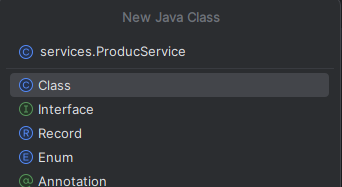


Not we can create one or more **Beans** . So now when we create **Beans** means object Creation , destruction ( **Full lifecycle** ) everything manage by Spring

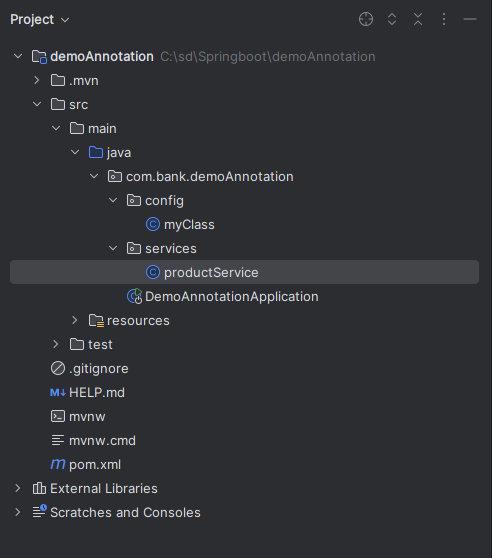
# Create Service Class ProductService in IntelliJ



Type **services.ProductService** in class name

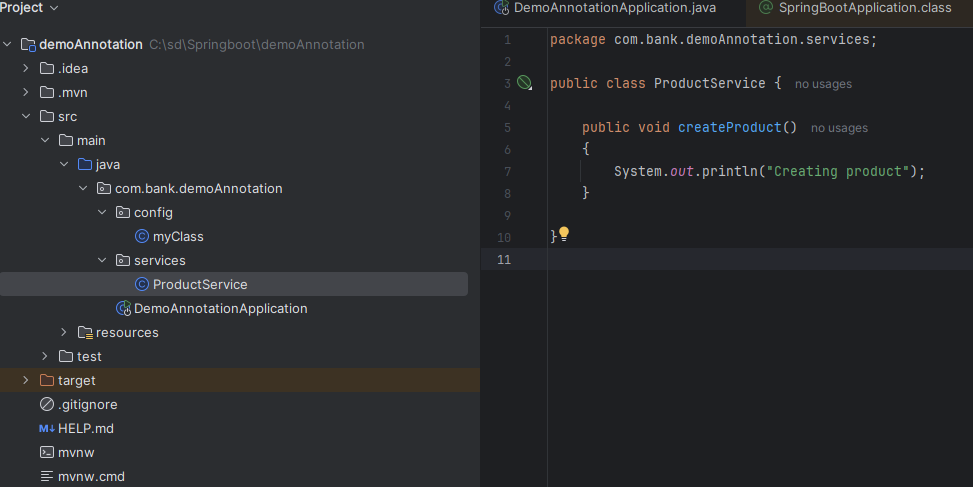


Class Created



Now Lets create a simple method

package com.bank.demoAnnotation.services;  
  
public class ProductService {  
  
 public void createProduct()  
 {  
 System.*out*.println("Creating product");  
 }  
  
}

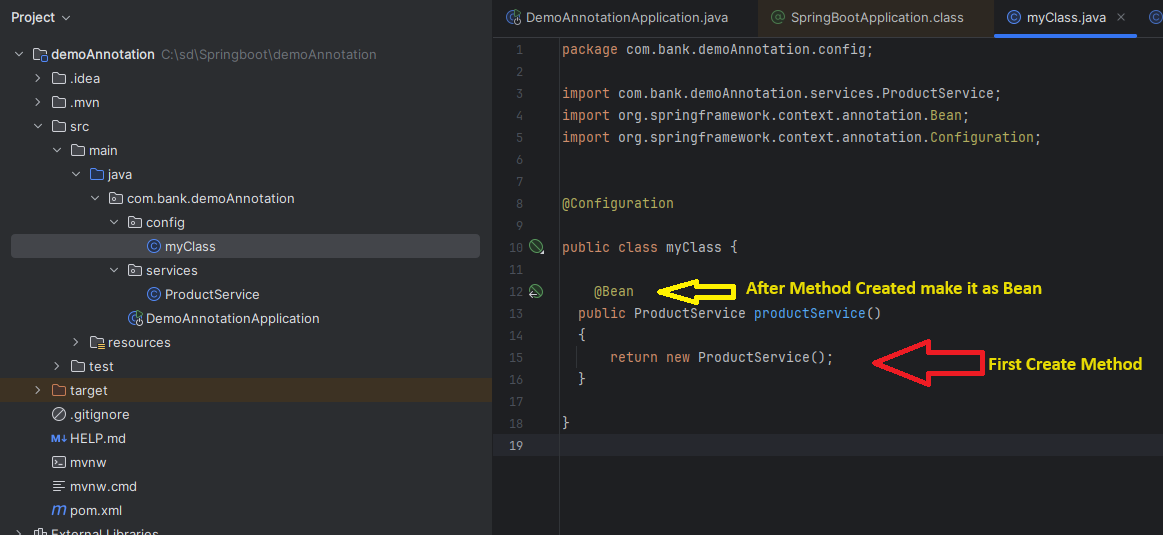


# Create Bean for Class productService under config class in IntelliJ

Now we want that the object of this class should be managed by Spring Container.

Open **myClass** under **config** Package and declare method **ProductService**() and make it as Beans by Annoting it with **@Beans**

@Bean  
public ProductService productService()  
{  
 return new ProductService();  
}

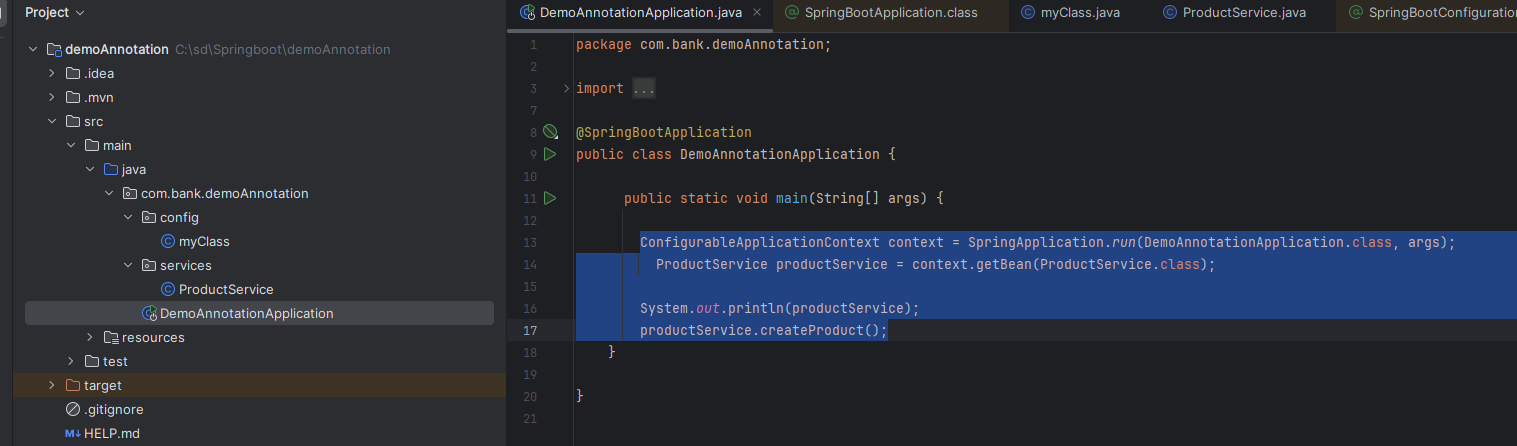


# Call Bean Class inside Main Application

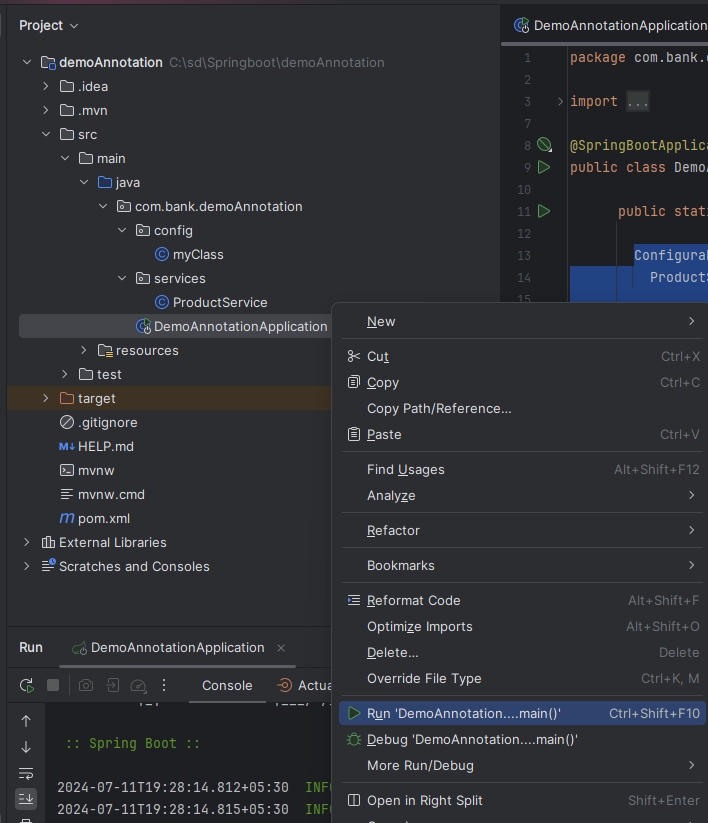
Now in main method we demand Beans we created in previous 8

Enter new line Main application

ConfigurableApplicationContext context = SpringApplication.*run*(DemoAnnotationApplication.class, args);  
 ProductService productService = context.getBean(ProductService.class);  
  
System.*out*.println(productService);  
productService.createProduct();



Now Run the program to Test



You can see the output ( Main class called Build class )

