

**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](#_Toc171585459)

[1. . Traditional Object Creation in Java 3](#_Toc171585460)

[2. Object Creation in Spring Boot 4](#_Toc171585461)

[2. Creating a Spring Boot Project 4](#_Toc171585462)

[3. Import Spring Boot Project in IntelliJ 6](#_Toc171585463)

[4. Create simple dev class in IntelliJ 9](#_Toc171585464)

[5. Call Dev Class inside Main Application (Java Method) 11](#_Toc171585465)

[6. Call Dev Class inside Main Application (Springboot Application Context Method) 13](#_Toc171585466)

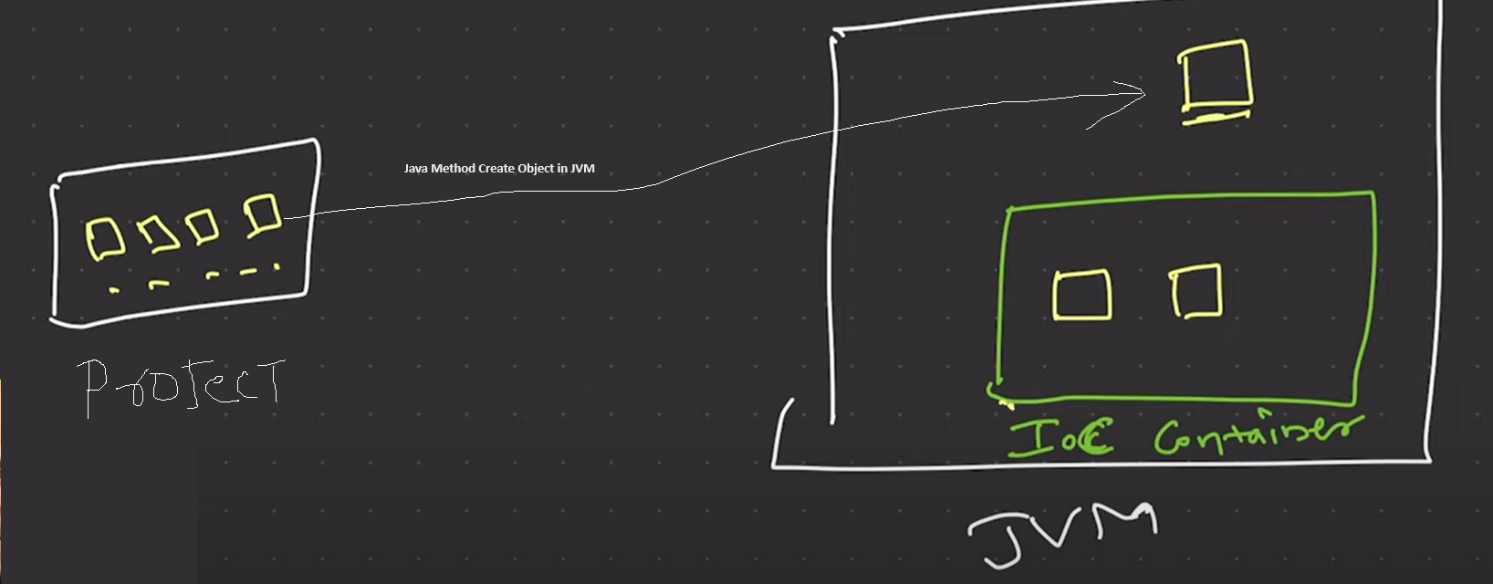
# Objective

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**

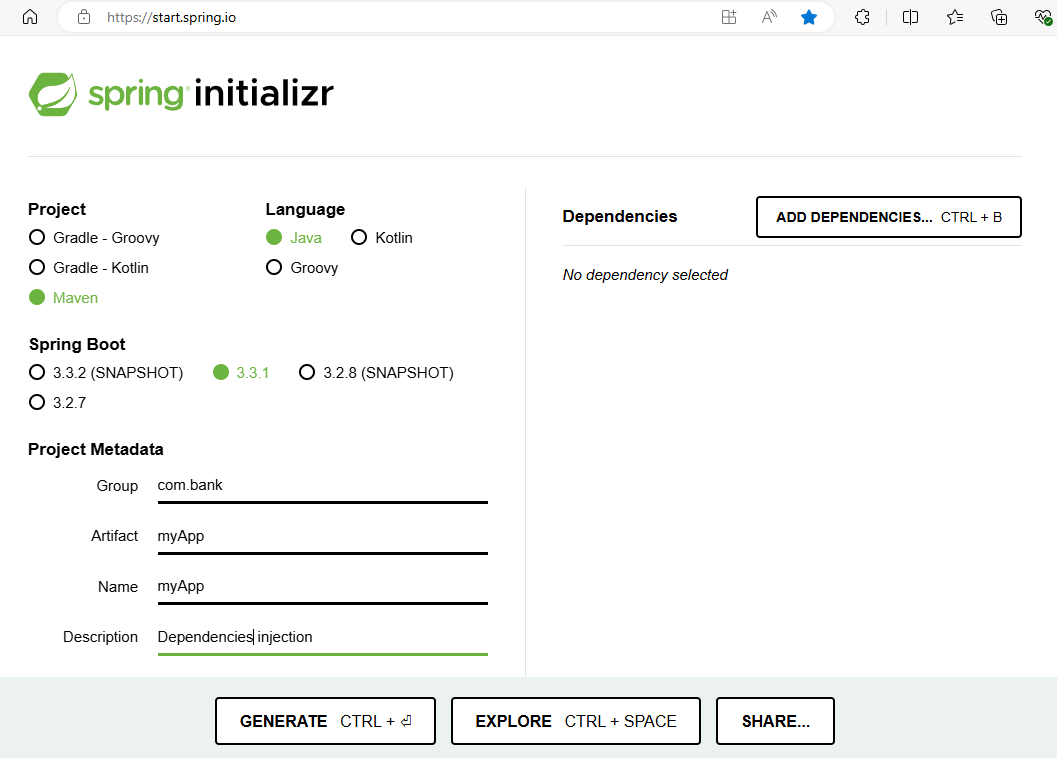
**Group** : com.bank

**Artifact** : myApp

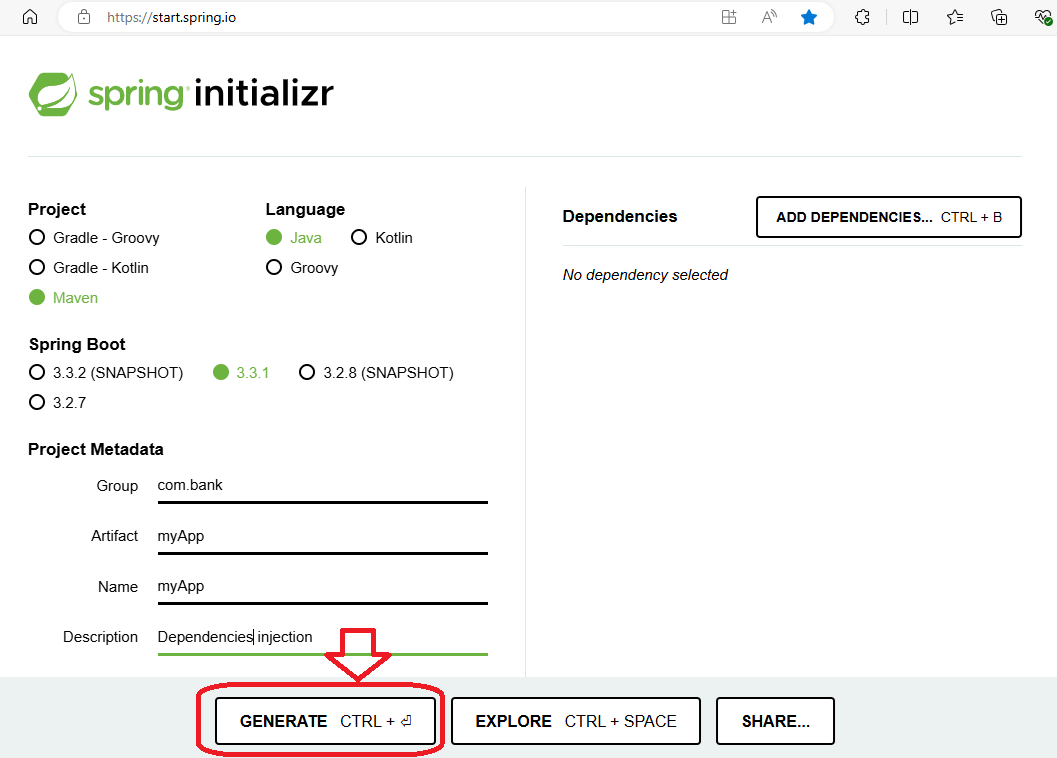
**Name** : myApp

**Description** : Dependencies Injection

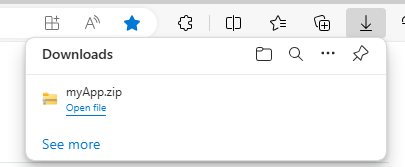
**PackageName** : com.bank.myApp



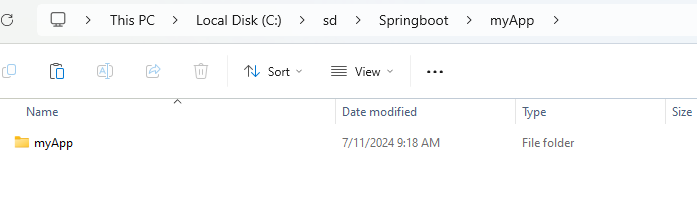
**Step 3: Click on GENERATE to download the code**



**Step 10:** Code should download now

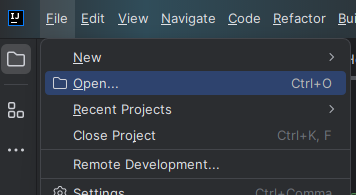


**Step 11: Extract** the project to your Project directory

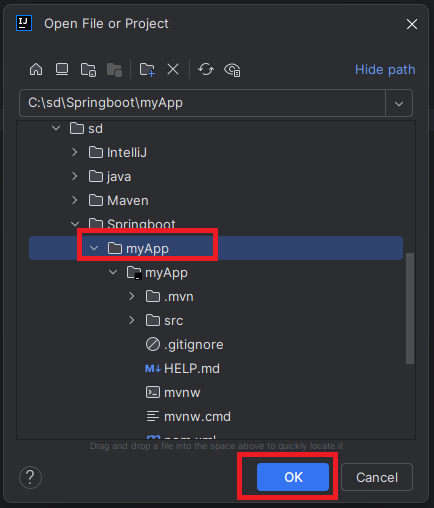


# Import Spring Boot Project in IntelliJ

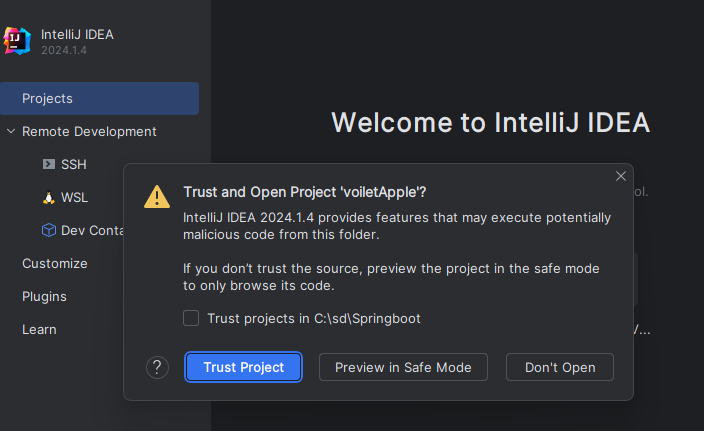
* Click File🡪 Open project

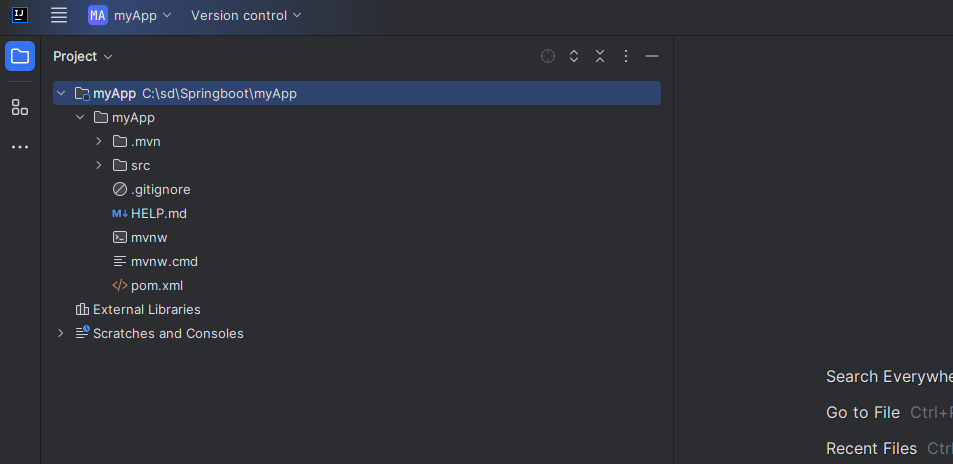


* Select your extract project and click **OK**

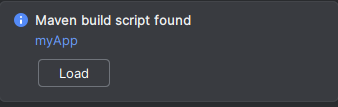


* If ask click **Trust project**

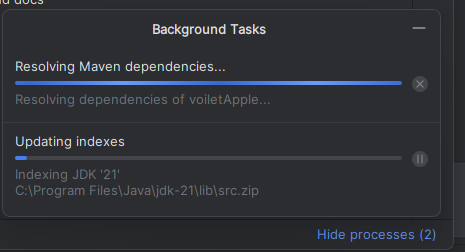




* If ask **Load** the Maven Build Script

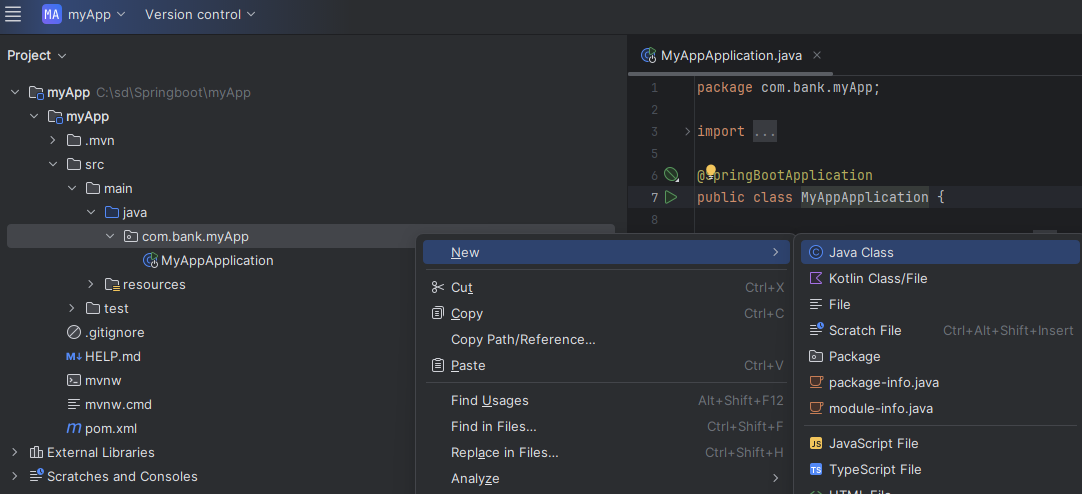


* It may takes couple of Minutes to download and Load the library

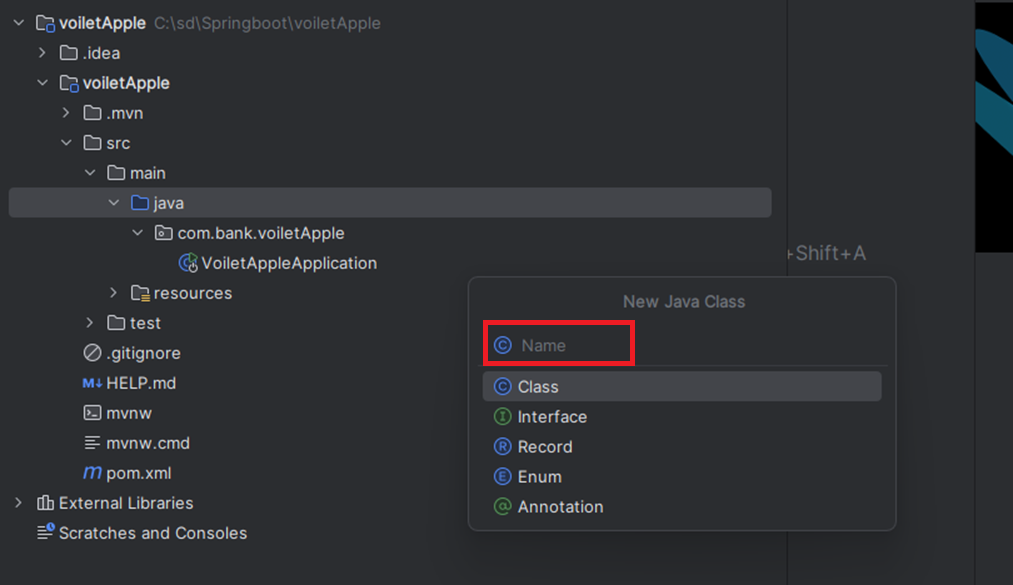


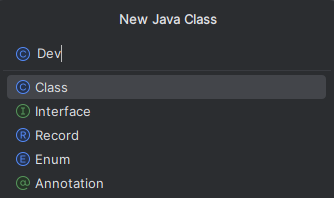
# Create simple dev class in IntelliJ

* Right Click the Application –New—Java Class

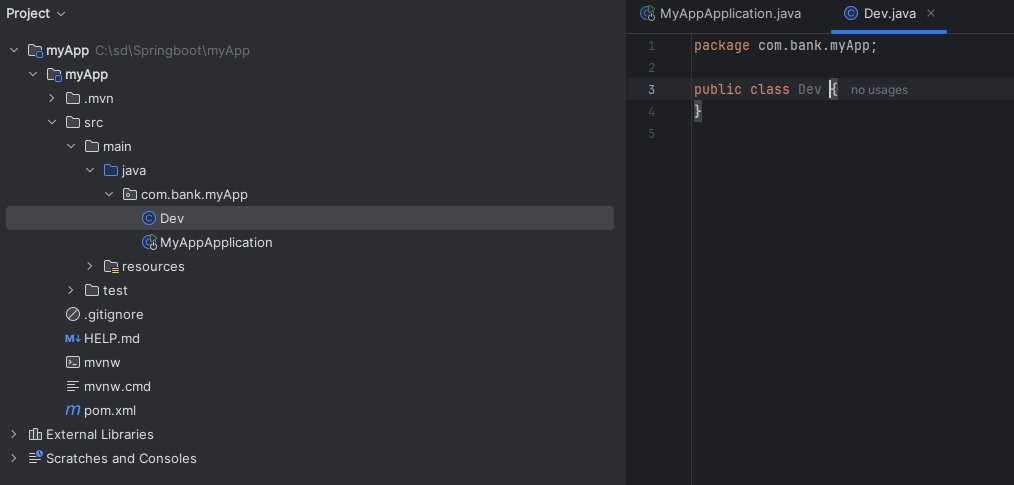


* Type **Dev** in Class Name and Press **enter** to create the class



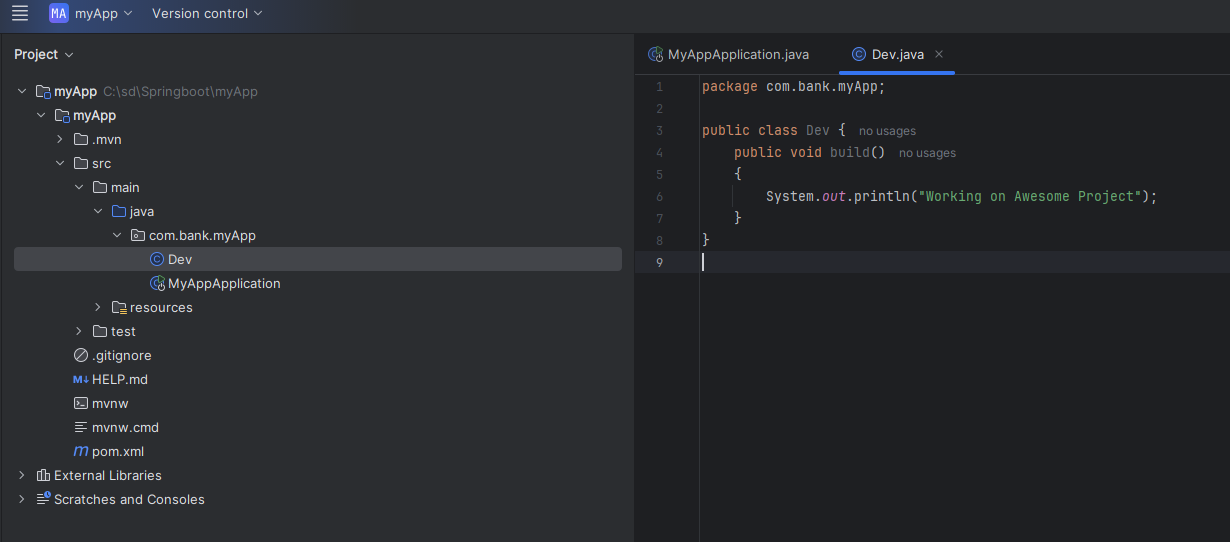


* Note the **class** created



* Type Below Code to Print Hello World

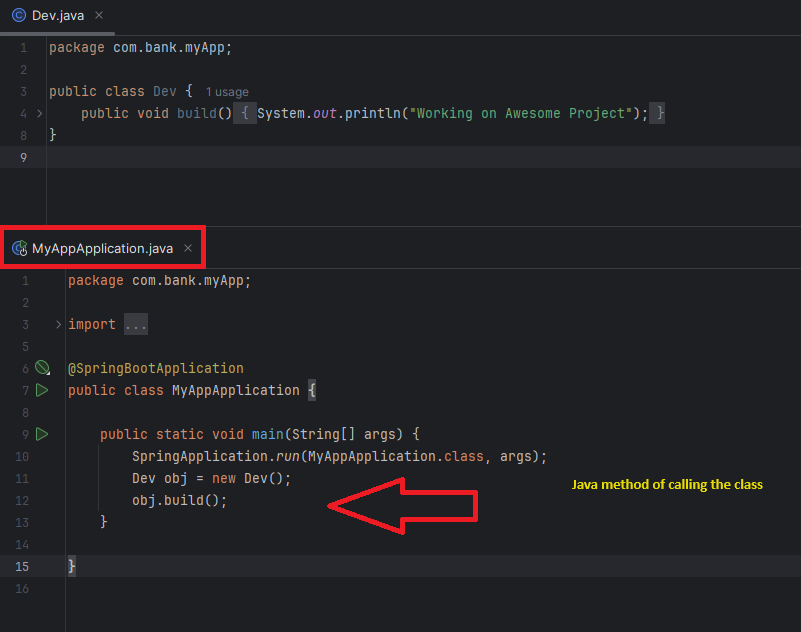
package com.bank.myApp;  
  
public class Dev {  
 public void build()  
 {  
 System.*out*.println("Working on Awesome Project");  
 }  
}



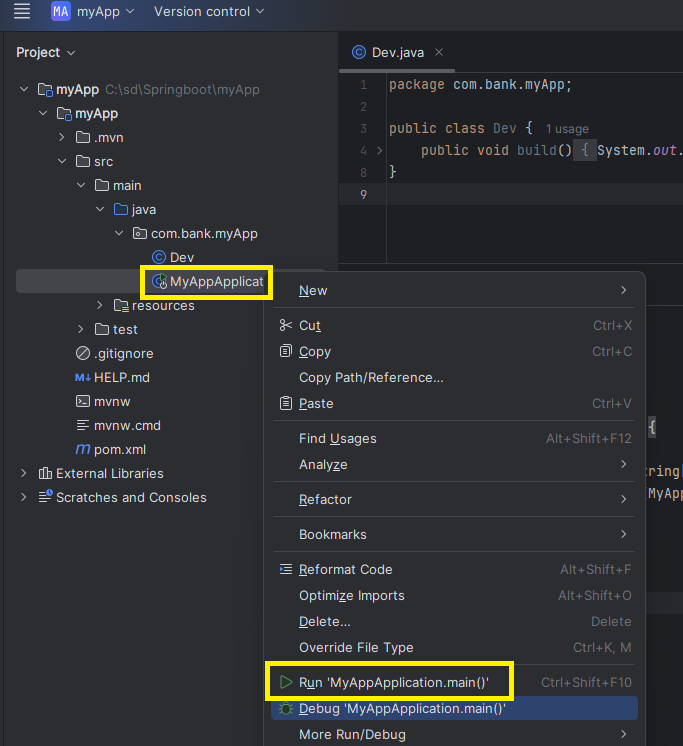
# Call Dev Class inside Main Application (Java Method)

Enter new line Main application

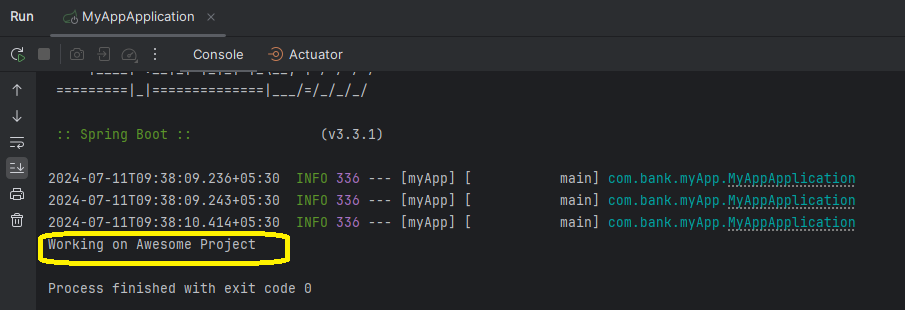
Dev obj = new Dev();  
obj.build();



Now Run the program to Test



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



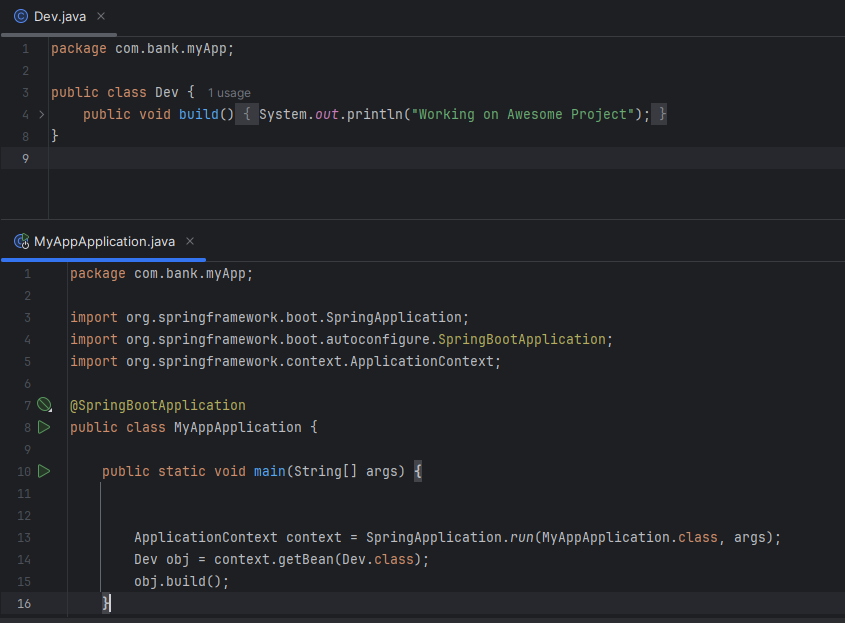
# Call Dev Class inside Main Application (Springboot Application Context Method)

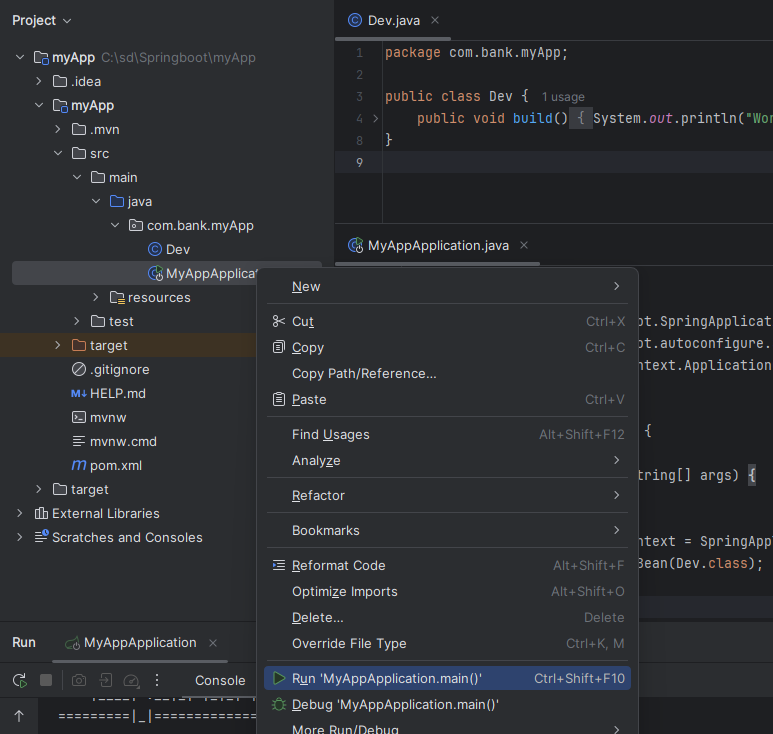
Here we assume that object of the class dev is already there inside container

package com.bank.myApp;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
  
@SpringBootApplication  
public class MyAppApplication {  
  
 public static void main(String[] args) {  
  
  
 **ApplicationContext context** = SpringApplication.*run*(MyAppApplication.class, args);  
 **Dev obj = context.getBean(Dev.class);  
 obj.build();** }

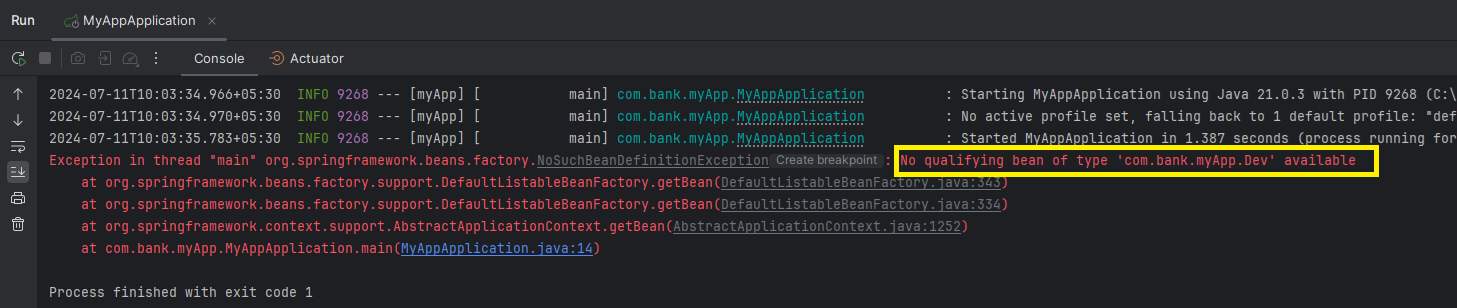
 **SpringApplication.run(MyAppApplication.class, args):** This line starts the Spring Boot application. It sets up the Spring environment, starts the Spring container, and performs auto-configuration.

 **ApplicationContext context:** This is an interface representing the Spring container. It holds all the Spring beans (objects) and manages them.





You Get the error that object build is not found



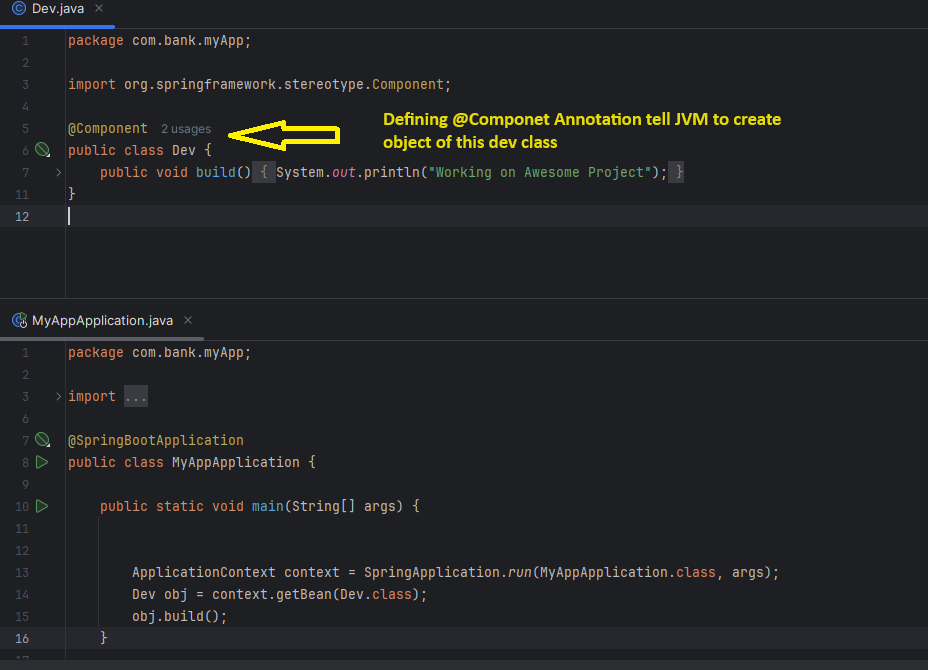
Spring Boot did not automatically create an **object (bean)** of the Dev class inside its container. Spring Boot doesn't create objects for every class automatically to avoid overloading the JVM, especially if there are a large number of classes.

In a typical Spring Boot application:

1. **Selective Object Creation**: Spring Boot only creates objects (beans) for classes that are specifically marked with annotations like **@Component, @Service, @Repository**, or **@Controller**. This is to ensure that the application only manages and maintains the objects that are actually needed, keeping resource usage efficient.
2. **Manual Annotation Required**: If you want Spring Boot to manage a class, you must annotate it appropriately. For instance, if Dev is a class you want managed by Spring, you should annotate it with **@Component** or another suitable stereotype annotation.

Here's how you can update your Dev class to be managed by Spring:

By adding the **@Component** annotation, you tell Spring to create and manage an instance of the Dev class. Then, you can retrieve it from the Spring container in your main application class.



Now Rerun it will show the correct output

