# Spring Boot Complete Tutorial - Master Class

<https://www.youtube.com/watch?v=zvR-Oif_nxg>

## Beans scopes

<https://docs.spring.io/spring-framework/reference/core/beans/factory-scopes.html>

GIT REPO DEV: <https://github.com/ok2073/springboot-dcb_tutorial.git>

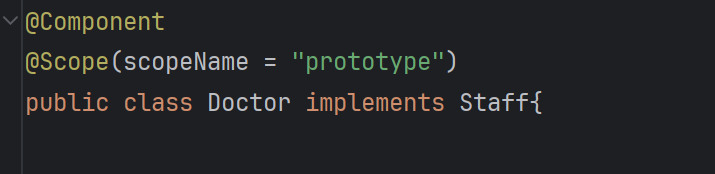
GIT REPO TUTORIAL: <https://github.com/shabbirdwd53/Springboot-Tutorial>

### Singleton

Only one shared instance of a singleton bean is managed, and all requests for beans with an ID or IDs that match that bean definition result in that one specific bean instance being returned by the Spring container.

### Prototype

The non-singleton prototype scope of bean deployment results in the creation of a new bean instance every time a request for that specific bean is made. That is, the bean is injected into another bean or you request it through a getBean() method call on the container. As a rule, you should use the prototype scope for all stateful beans and the singleton scope for stateless beans.



### Request, Session, Application, and WebSocket Scopes

The request, session, application, and websocket scopes are available only if you use a web-aware Spring ApplicationContext implementation (such as XmlWebApplicationContext). If you use these scopes with regular Spring IoC containers, such as the ClassPathXmlApplicationContext, an IllegalStateException that complains about an unknown bean scope is thrown.

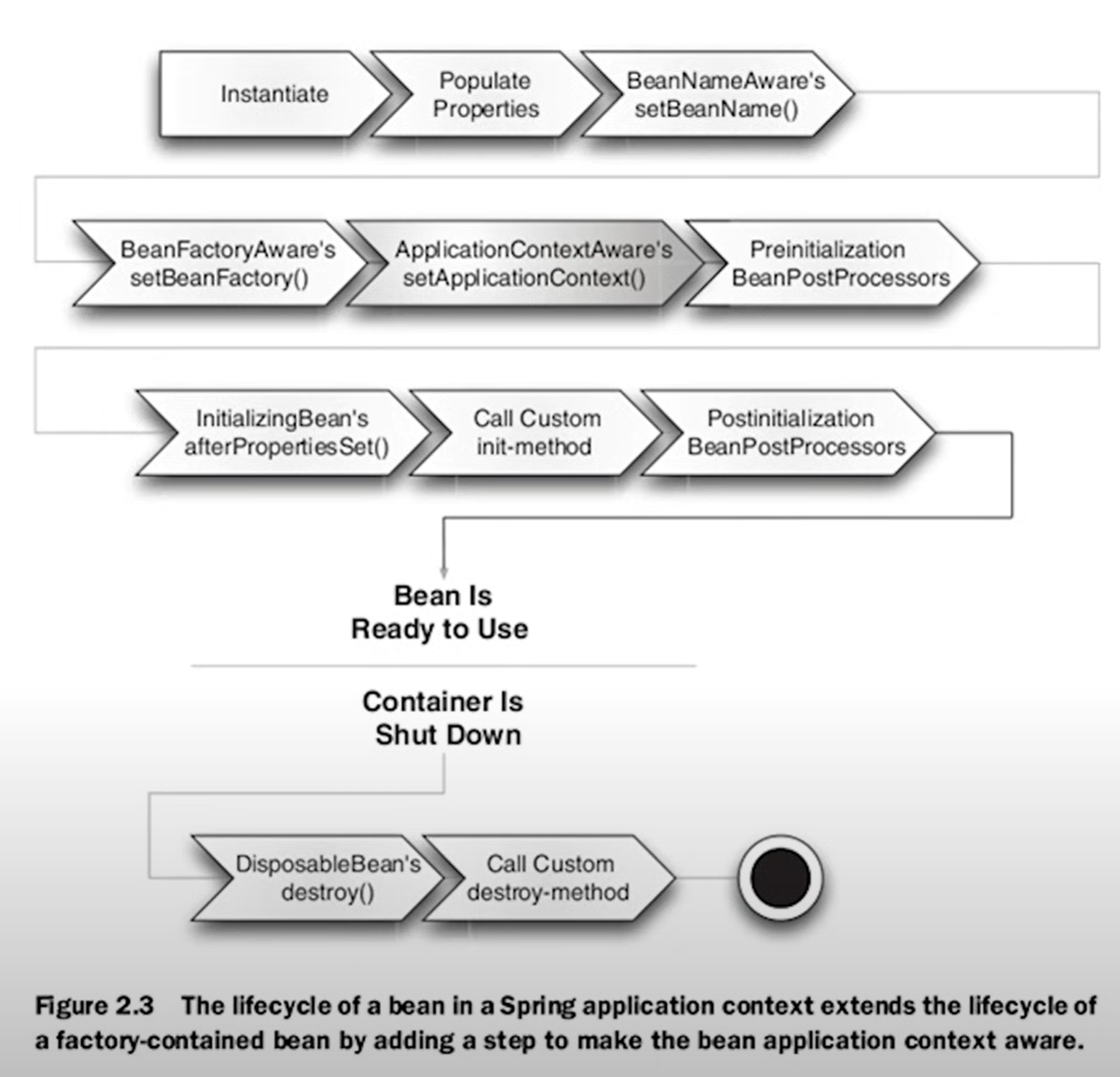
Request: Scopes a single bean definition to the lifecycle of a single HTTP request.

Session: Scopes a single bean definition to the lifecycle of an HTTP session.

Application: Scopes a single bean definition to the lifecycle of a ServletContext.

Websocket: Scopes a single bean definition to the lifecycle of a WebSocket.

### Beans Lifecycle



## Spring AOP

Aspect-oriented programming (AOP) is a programming paradigm that aims to modularize cross-cutting concerns, which are functionalities that affect multiple parts of a software application. It allows the separation of such concerns from the main application logic, promoting better modularity and reusability. AOP achieves this by creating entities called "aspects" that encapsulate these cross-cutting concerns and can be applied across different parts of the application.

i.e.

* Logging
* Authorization and authentication

## Spring Boot

Spring is a Java framework built to create enterprise ready applications.

Spring Boot is an extension layer of Spring framework that simplifies the development of microservices and web applications. It is designed to create stand-alone, production-ready applications with minimal Spring configuration, allowing developers to "just run" their applications.

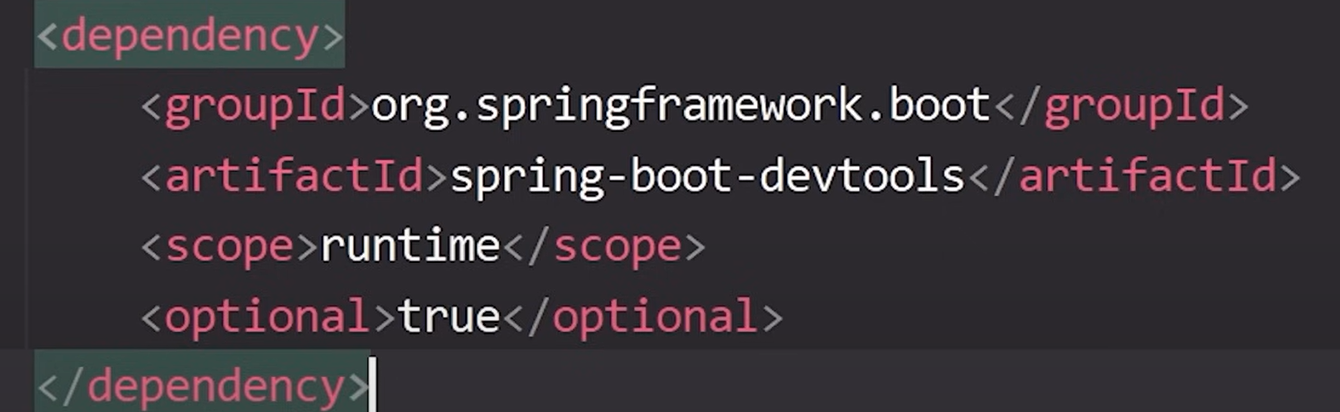
**Inversion of Control (IoC) is a design principle** where the control flow of a program is inverted. Instead of the programmer controlling the flow, external sources such as frameworks, services, or other components take control of it.

**Dependency Injection (DI) is a specific pattern used to implement IoC**. It is a way of providing objects with their dependencies, rather than the object creating or looking for its dependencies. This can be achieved through various methods such as constructor injection, setter/getter injection, interface injection, or service locator.

### Running a Spring Boot application from terminal



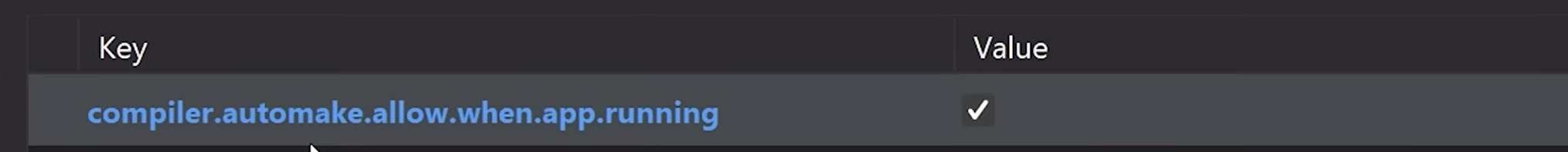
### Spring Boot Dev tools



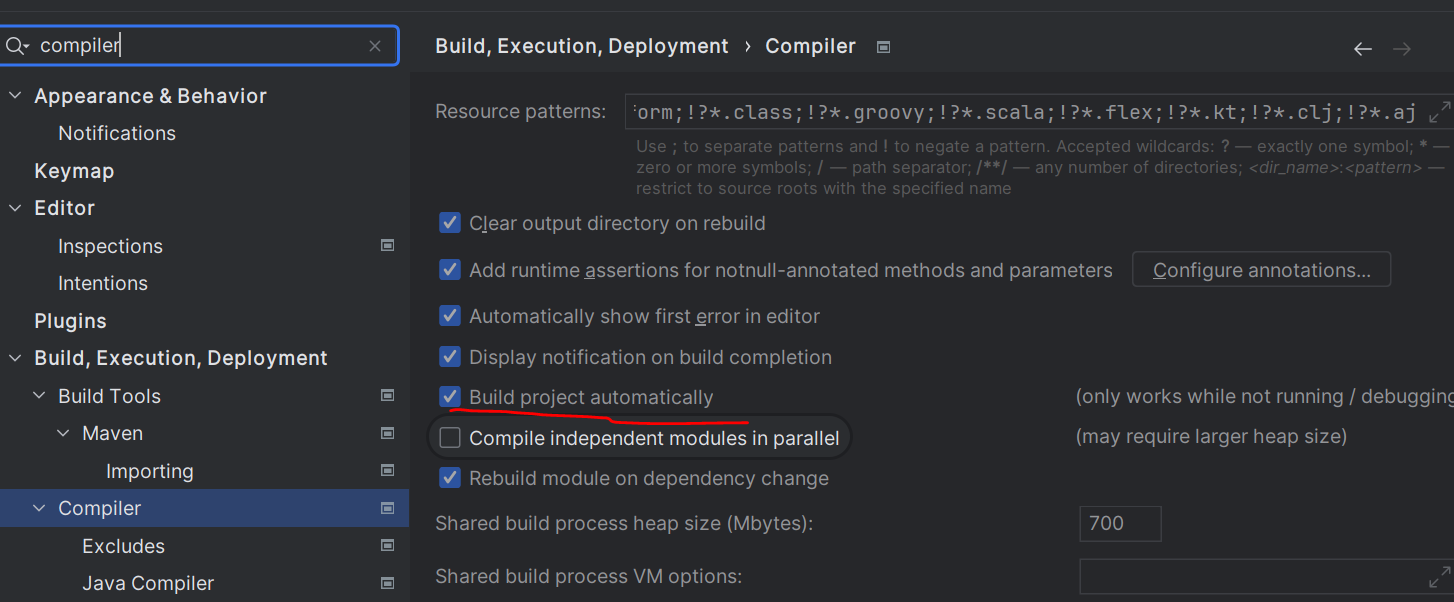
CTRL + SHIFT + A

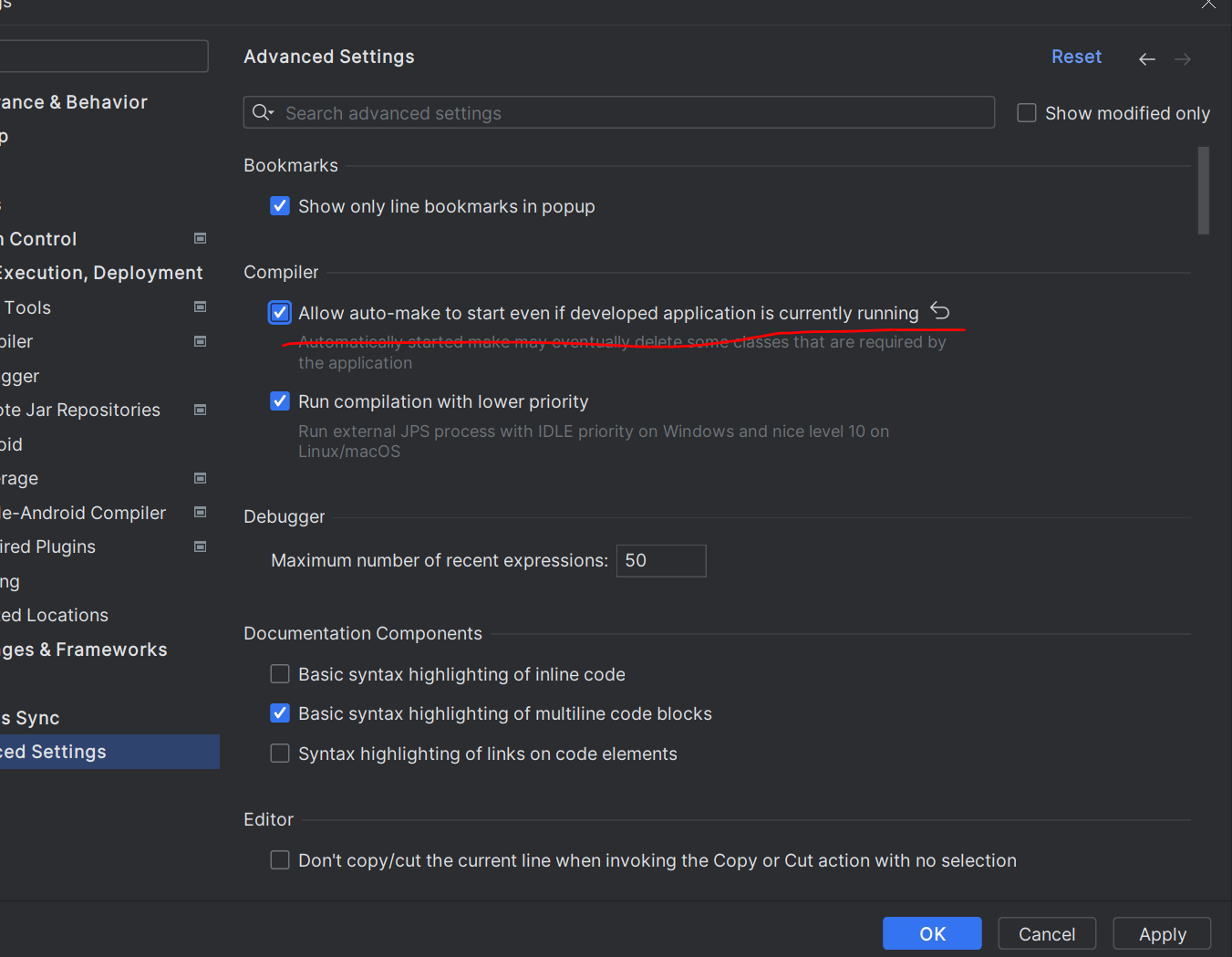
Registry

And search



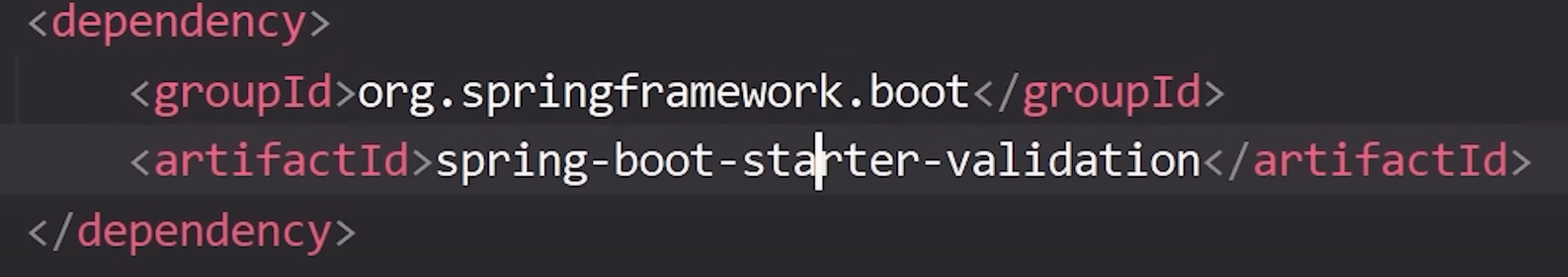
Then Settings/Compiler

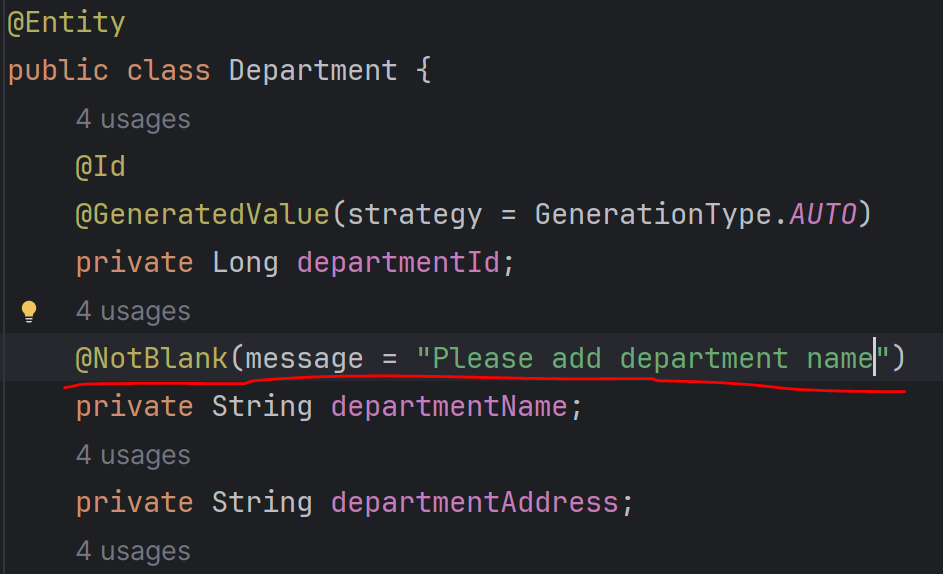




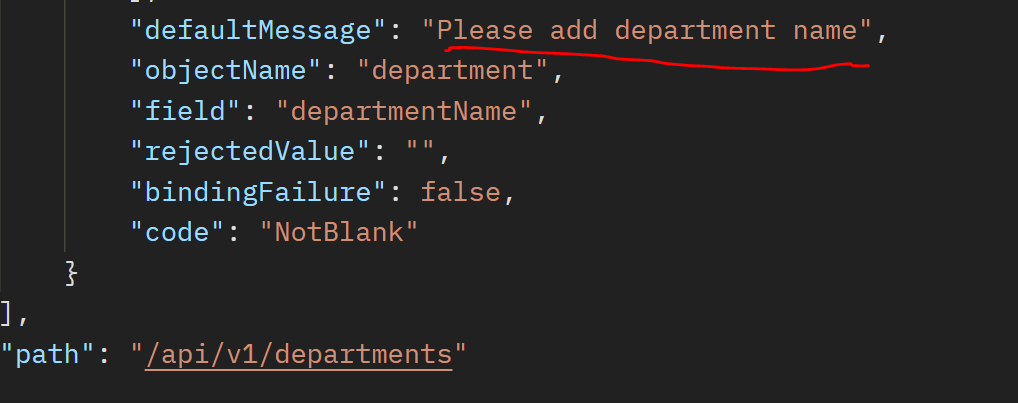
### Hibernate validator

It is inside:

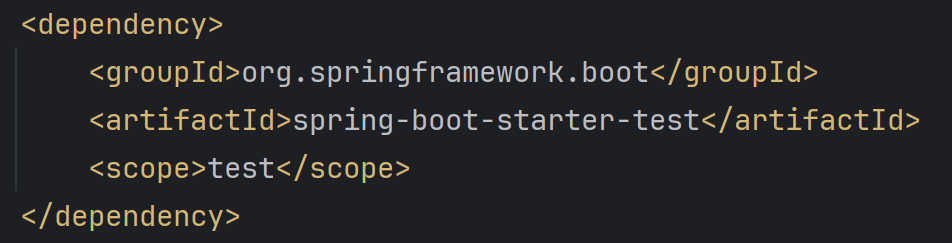








### Unit testing



### Adding config in properties files



A screen shot of a computer program

Description automatically generated

### Adding application.yml

A screenshot of a computer

Description automatically generated

### Spring Boot profiles

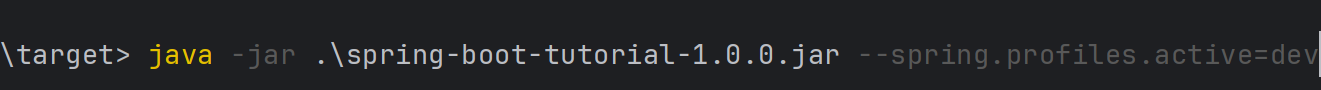
A screenshot of a computer program

Description automatically generated

--- define a new document



Running profiles from command line:



### Actuator

4.16