# **Lab: Spring Core**

## **Prerequisite - project setup**

### **Create a new Spring Boot project**

You can use [Spring Initializr](https://start.spring.io/) with the following setup:

**Project:** Maven or Gradle

**Language:** Java

**Spring Boot Version:** Default stable version, don't use SNAPSHOT version

**Group:** **bg.softuni**

**Artifact + Name:** **spring-core**

**Description:** Spring Core Exercise

**Package name:** bg.softuni.spring-core  
**Packaging**: Jar

**Java:** 17

**Dependencies:** *No dependencies required*

Картина, която съдържа текст, екранна снимка, софтуер, номер

Генерираното от ИИ съдържание може да е неправилно.

### **Import the project into your IDE and run the application**

After starting, the application will automatically shut down (**this is expected**).

Your console output should look similar to the following example:

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Генерираното от ИИ съдържание може да е неправилно.

## **Exercise 1: Defining Beans**

Define two simple classes that represent services in your application: **GreetingService** and **AddressService**. Each of them should expose a single method:

* **GreetingService#getGreeting()** – returns a greeting message such as "**Hello User, welcome to the Spring Core Lab!**".
* **AddressService#getAddress()** – returns a string such as "**My address is Sofia, Bulgaria.**".

Your goal is to register both classes as **Spring-managed beans** using the appropriate annotation. Once registered, retrieve the two beans from the Application Context in the main() method of your application, and invoke their methods in the following order:

1. Print the result of **GreetingService#getGreeting()**
2. Print the result of **AddressService#getAddress()**

### **Expected Output**

Hello User, welcome to the Spring Core Lab!

My address is Sofia, Bulgaria.

**Example Console Look**  
Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Генерираното от ИИ съдържание може да е неправилно.

## **Exercise 2: Dependency Injection**

Extend the functionality of your existing **GreetingService** by introducing a dependency to the other service in the project. Your task is to restructure **GreetingService** so that it depends on **AddressService** in order to produce a full greeting message that includes **both** a **welcome phrase** and **an address**.

Update the implementation of the method **getGreeting()** to make it return the greeting message **and** the address **on the same line**.

**Important:** You should **only retrieve and use the GreetingService bean** in your application's **main()** method. The call to **getGreeting()** must return the full message. **Do not retrieve or call AddressService directly from main()**.

**Expected output**

Hello User, welcome to the Spring Core Lab! My address is Sofia, Bulgaria.

## **Exercise 3: External Configuration**

To make your application more flexible and adaptable, you should no longer rely on hardcoded values inside your **service classes**. Instead, extract the user name and address into configuration properties (application.properties), and **inject** them into your services using Spring's **@Value** annotation.

Define two new properties:

* **user.name** with value **Peter Parker**
* **user.address** with value **Queens, New York**

Your task is to inject these values into your service classes:

* In **GreetingService**, inject **user.name** using **@Value**
* In **AddressService**, inject **user.address** using **@Value**

Update the methods in both services to return messages that incorporate the injected values instead of hardcoded strings. The method **getGreeting()** should return the full message, including both the greeting and the address - just like in the previous task, but now dynamically assembled using configuration values.

**Important:** Make sure no literal names or addresses are present in the Java code. The message must reflect what's currently set in the configuration file.

**Expected Output**

Hello **Peter Parker**, welcome to the Spring Core Lab! My address is **Queens, New York**.

## **Exercise 4: Magic Bean**

Create a class of your choice (e.g. **MagicBean**) and ensure that when you print the object returned from **ApplicationContext#getBean(...)**, the output contains different characters after the **@** symbol each time the bean is retrieved and printed.

For example, calling the following line multiple times:

**System.out.println(context.getBean(MagicBean.class));**

This should result in output such as:

bg.softuni.spring\_core.MagicBean**@3b1f2a7e** bg.softuni.spring\_core.MagicBean**@77f99a05** *(a different value after the* ***@*** *then the one printed earlier)*

Your task is to **find a way to achieve this behavior using Spring’s configuration capabilities**. You are **not allowed** to manually instantiate the object yourself (e.g. with new). Everything must go through the Spring context.