- **Name:** hello-springboot
- **Description:** Hello World Spring Boot Application
- **Package Name:** com.example.springboot.app
- **Packaging:** Jar
- **Java:** Choose Java 17 or higher (recommended for modern Spring Boot).

3. **Add Dependencies:** In the "Dependencies" section, search for and add the following:
   - **Spring Web:** This starter pulls in Spring Core, Spring MVC, and an embedded Tomcat server, which is perfect for our simple "Hello World" web application (even though we won't expose a web endpoint, it's a common starter).
   - **Lombok:** (Optional but highly recommended) Reduces boilerplate code for getters, setters, constructors, etc.
4. **Generate and Download:** Click the "Generate" button. This will download a .zip file containing your new Spring Boot project.
5. **Import into IDE:** Unzip the downloaded file and import the project into your favorite IDE (IntelliJ IDEA, Eclipse, VS Code with Java extensions). Your IDE will typically recognize it as a Maven project and set it up automatically.
   - You will find a main application class already created for you, typically named HelloSpringbootApplication.java in the com.example.springboot.app package.

STEP 2: Create a Simple Message Service

We'll create the same simple service as before that provides a "Hello World" message.

- Create a new package com.example.springboot.service in src/main/java/ (if it doesn't already exist).
- Create a Java class named MessageService.java inside this package:

```
// src/main/java/com/example/springboot/service/MessageService.java
package com.example.springboot.service;

import org.springframework.stereotype.Service; // Import the @Service annotation

/**
 * This class is a simple service that provides a greeting message.
 * The @Service annotation tells Spring that this is a service component
```

```
 * and should be managed by the Spring IoC container.
 * Spring Boot's auto-configuration and component scanning will automatically
 * detect and register this as a bean.
 */
@Service // Marks this class as a Spring service component
public class MessageService {

    /**
     * Returns a simple "Hello, Spring Boot World!" message.
     * @return The greeting message.
     */
    public String getMessage() {
        return "Hello, Spring Boot World!";
    }
}
```

STEP 3: Update the Main Spring Boot Application Class

Spring Boot automatically generates a main class with @SpringBootApplication. We'll modify its main method to retrieve and use our MessageService.

- Open the HelloSpringbootApplication.java file (or whatever your main application class is named, typically in com.example.springboot.app package).+

- Modify it as follows:

```
// src/main/java/com/example/springboot/app/HelloSpringbootApplication.java
package com.example.springboot.app;

import com.example.springboot.service.MessageService; // Import our service class
import org.springframework.boot.SpringApplication; // Import SpringApplication
import org.springframework.boot.autoconfigure.SpringBootApplication; // Import
@SpringBootApplication
import org.springframework.context.ConfigurableApplicationContext; // Import
ConfigurableApplicationContext

/**
 * The main entry point for our Spring Boot application.
 * @SpringBootApplication is a convenience annotation that adds:
 * - @Configuration: Tags the class as a source of bean definitions.
```

```
 * - @EnableAutoConfiguration: Tells Spring Boot to start adding beans based on classpath settings,
 * other beans, and various property settings.
 * - @ComponentScan: Tells Spring to look for other components, configurations, and services
 * in the 'com.example.springboot.app' package and its sub-packages.
 */
@SpringBootApplication
public class HelloSpringbootApplication {

    public static void main(String[] args) {
        System.out.println("--- Starting Spring Boot Application ---");

        // 1. Run the Spring Boot application.
        // This method bootstraps the application, creates the Spring ApplicationContext,
        // performs auto-configuration, and starts any embedded servers (if applicable).
        ConfigurableApplicationContext context = SpringApplication.run(HelloSpringbootApplication.class, args);

        System.out.println("\n--- Retrieving and Using Spring Bean ---");
        // 2. Retrieve the MessageService bean from the container.
        // Spring Boot's @ComponentScan (part of @SpringBootApplication)
        // automatically found and registered our MessageService.
        MessageService messageService = context.getBean(MessageService.class);

        // 3. Use the retrieved bean to get the message.
        String message = messageService.getMessage();
        System.out.println("Received message from Spring Boot bean: " + message);

        // 4. Close the application context gracefully.
        // In a real web application, the server would keep running.
        // For a simple command-line app, we close the context.
        context.close();
        System.out.println("\n--- Spring Boot Application Shut Down ---");
        System.out.println("Congratulations! You've successfully run your first Spring Boot application!");
    }
}
```

STEP 4: Run the Spring Boot Application

Running a Spring Boot application is incredibly simple.

1. **Run from your IDE:**
    - Open the HelloSpringbootApplication.java file in your IDE.
    - Right-click within the file and select "Run 'HelloSpringbootApplication.main()'" or click the green play button next to the main method.
2. **Run from Terminal (Maven Spring Boot Plugin):**
    - Open your terminal or command prompt.
    - Navigate to the root directory of your hello-springboot project (where pom.xml is located).
    - Run the Spring Boot Maven plugin command:
      mvn spring-boot:run
    - This command will compile your project, package it, and then run the Spring Boot application.

You should see output similar to this in your console (Spring Boot's logs will be more verbose, but you'll find your custom messages within them):

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.x.x)

... (Spring Boot startup logs) ...

--- Starting Spring Boot Application ---

--- Retrieving and Using Spring Bean ---
Received message from Spring Boot bean: Hello, Spring Boot World!

--- Spring Boot Application Shut Down ---
Congratulations! You've successfully run your first Spring Boot application!

... (Spring Boot shutdown logs) ...
```

This demonstrates how Spring Boot automates much of the configuration and setup, allowing you to focus directly on your application's logic.

Activity 2.1: Spring DI Fundamentals

This activity will guide you through building a simple Spring application to understand Dependency Injection (DI), Spring's Inversion of Control (IoC) container, component scanning, autowiring, and bean scopes using Java-based configuration.

Wiring Beans (Annotations & Java Config)

"Wiring" refers to the process of connecting beans with their dependencies. Spring provides several ways to do this. We will focus on **Annotations** and **Java Configuration**.

STEP 1: Project Setup (Maven)

1. **Create a New Maven Project:**
   - Open your IDE (IntelliJ IDEA, Eclipse, VS Code).
   - Create a new Maven project.
   - Set GroupId to com.example.di and ArtifactId to di-demo.
   - Package : com.example.di.demo
2. **Update pom.xml:**
   - Add the spring-context dependency.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example.di</groupId>
    <artifactId>di-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <spring.version>6.1.0</spring.version> <!-- Use a recent stable Spring version
-->
    </properties>

    <dependencies>
        <!-- Spring Core and Context for DI -->
```