```
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
    </dependencies>
</project>
```

3. **Download Dependencies:**
    - Right click on project and locate Maven option -> Click Sync Project to download maven dependencies (ntelliJ IDEA)
    - Option may be different for other IDE
    - Or use “mvn clean install” command using terminal in Project home folder.

STEP 2: Create a Simple Message Service

We'll create a simple service that provides a "Hello World" message.

- Create a new package com.example.spring.service in src/main/java/.
- Create a Java class named MessageService.java inside this package:

```
// src/main/java/com/example/spring/service/MessageService.java
package com.example.spring.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 will automatically create an instance (bean) of this class.
 */
@Service // Marks this class as a Spring service component
public class MessageService {

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

```
    }
}
```

STEP 3: Create Spring Configuration

We need a configuration class to tell Spring where to find our components (like MessageService).

- Create a new package com.example.spring.config in src/main/java/.
- Create a Java class named AppConfig.java inside this package:

```
// src/main/java/com/example/spring/config/AppConfig.java
package com.example.spring.config;

import org.springframework.context.annotation.ComponentScan; // Import @ComponentScan
import org.springframework.context.annotation.Configuration; // Import @Configuration

/**
 * This class is the Spring configuration for our application.
 * @Configuration indicates that this class contains bean definitions.
 * @ComponentScan tells Spring to scan the specified base package
 * (and its sub-packages) for Spring-annotated components like @Service,
 * @Component, @Repository, @Controller, etc., and register them as beans.
 */
@Configuration // Marks this class as a Spring configuration class
@ComponentScan(basePackages = "com.example.spring") // Scans for components in "com.example.spring" and its sub-packages
public class AppConfig {
    // No explicit @Bean methods are needed here for MessageService
    // because @ComponentScan will find @Service MessageService automatically.
}
```

STEP 4: Create the Main Application Class

This is where we'll start the Spring IoC container and retrieve our MessageService bean.

- Create a new package com.example.spring.app in src/main/java/.

- Create a Java class named HelloWorldApp.java inside this package:

```
// src/main/java/com/example/spring/app/HelloWorldApp.java
package com.example.spring.app;

import com.example.spring.config.AppConfig; // Import our configuration class
import com.example.spring.service.MessageService; // Import our service class
import org.springframework.context.annotation.AnnotationConfigApplicationContext; //
Import Spring's application context

/**
 * The main application class to demonstrate a basic Spring "Hello World".
 * It initializes the Spring IoC container and retrieves a bean to use it.
 */
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("--- Starting Spring IoC Container ---");

        // 1. Create a Spring IoC Container (ApplicationContext)
        // AnnotationConfigApplicationContext is used to load Java-based configurations
(@Configuration classes).
        // When 'AppConfig.class' is passed, Spring reads it, performs component
scanning,
        // and creates all the necessary beans (including MessageService).
        try (AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class)) {

            System.out.println("\n--- Retrieving and Using Spring Bean ---");
            // 2. Retrieve the MessageService bean from the container.
            // Spring provides the instance of MessageService that it created and managed.
            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 bean: " + message);

        } // The 'try-with-resources' ensures the context is closed automatically.
        System.out.println("\n--- Spring IoC Container Shut Down ---");
        System.out.println("Congratulations! You've successfully run your first Spring
application!");
    }
```

}

How to Run the Application:

1. **Run from your IDE:**
    - Open the HelloWorldApp.java file in your IDE.
    - Right-click within the file and select "Run 'HelloWorldApp.main()'" or click the green play button next to the main method.

You should see output similar to this in your console:

--- Starting Spring IoC Container ---

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

--- Spring IoC Container Shut Down ---
Congratulations! You've successfully run your first Spring application!

This output demonstrates that Spring successfully initialized its container, found your MessageService (due to @Service and @ComponentScan), created an instance of it, and provided it when you requested the bean. This is the essence of Dependency Injection and Inversion of Control!

Activity 1.2: Basic Setup & "Hello World" Application using Spring Boot

This activity will guide you through creating a simple "Hello World" application using Spring Boot, demonstrating how it simplifies project setup and execution compared to traditional Spring.

Introduction: What is Spring Boot?

Spring Boot is a project built on top of the Spring Framework that aims to simplify the development of production-ready Spring applications. It provides:

- **Auto-configuration:** Automatically configures your Spring application based on the dependencies you've added.
- **Embedded Servers:** Can embed servers like Tomcat, Jetty, or Undertow directly into your executable JAR, so you don't need to deploy WAR files.
- **Opinionated Defaults:** Provides sensible default configurations to reduce the need for manual setup.
- **Stand-alone Applications:** Allows you to create stand-alone applications that can be run with just java -jar.

In essence, Spring Boot helps you "just run" your Spring applications with minimal fuss.

STEP 1: Spring Boot Project Setup

The easiest and recommended way to create a Spring Boot project is by using **Spring Initializr**.

1. **Go to Spring Initializr:** Open your web browser and navigate to https://start.spring.io/.
2. **Configure Your Project:**
    - **Project:** Maven Project (or Gradle, if preferred)
    - **Language:** Java
    - **Spring Boot:** Choose the latest stable version (e.g., 3.x.x).
    - **Group:** com.example.springboot (or your preferred group ID)
    - **Artifact:** hello-springboot (or your preferred artifact ID)