**Tutorial: Introduction to Spring Core & Spring Boot**

**Overview**

Spring Framework is one of the most widely used Java frameworks for building enterprise applications. This tutorial provides a step-by-step introduction to **Spring Core** and **Spring Boot**, focusing on setting up a project and exploring key concepts.

**Step 1: Prerequisites**

Before diving in, ensure you have:

* **Java Development Kit (JDK)** (version 11 or later is recommended)
* **Maven** or **Gradle** installed
* **IDE** such as IntelliJ IDEA, Eclipse, or Visual Studio Code
* **Spring Boot CLI** (optional but helpful)

**Step 2: Understanding Spring Core**

Spring Core provides the foundational building blocks for the entire Spring ecosystem, such as dependency injection and inversion of control (IoC).

**Key Concepts:**

1. **Dependency Injection (DI)**:
   * The process by which Spring injects objects into other objects (dependencies) at runtime.
   * Enables loose coupling between components.
2. **Inversion of Control (IoC)**:
   * Refers to the control being inverted (shifted from the application code to the framework).
   * Spring IoC container manages the lifecycle and configurations of beans.

**Step 3: Setting Up a Basic Spring Core Application**

1. **Create a Maven/Gradle Project**:
   * Create a new project in your IDE.
   * Add the following dependencies to your pom.xml (Maven):

xml

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

1. **Define a Bean**:
   * Create a class and annotate it with @Component or configure it in an XML file.

java

@Component

public class GreetingService {

public String greet() {

return "Hello, Spring Core!";

}

}

1. **Use the ApplicationContext**:
   * Create a simple Java application to load the Spring IoC container and retrieve a bean.

java

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

GreetingService service = context.getBean(GreetingService.class);

System.out.println(service.greet());

}

}

**Step 4: Introduction to Spring Boot**

Spring Boot simplifies the development of Spring applications by providing auto-configuration and an embedded server. It eliminates boilerplate configurations.

**Step 5: Setting Up a Spring Boot Project**

1. **Create a Spring Boot Project**:
   * Use **Spring Initializr** (<https://start.spring.io/>) to generate a Spring Boot project.
   * Add dependencies such as:
     + Spring Web
     + Spring Boot DevTools
   * Download the project and import it into your IDE.
2. **Understand the Structure**:
   * src/main/java: Contains application source code.
   * src/main/resources: Contains configuration files like application.properties.
3. **Run the Spring Boot Application**:
   * Spring Boot applications include an embedded Tomcat server.
   * Run the main method in the Application class:

java

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**Step 6: Create a Simple REST API**

1. **Add a Controller**:

java

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class GreetingController {

@GetMapping("/greet")

public String greet() {

return "Hello, Spring Boot!";

}

}

1. **Test the Application**:
   * Start the application by running the DemoApplication class.
   * Visit http://localhost:8080/greet in your browser or use a tool like Postman.

**Step 7: Spring Boot Features**

1. **Auto-Configuration**:
   * Spring Boot automatically configures beans and settings based on the dependencies in your project.
2. **Embedded Server**:
   * No need to deploy your application to an external server. Use embedded Tomcat, Jetty, or Undertow.
3. **Actuator**:
   * Add the Actuator dependency to monitor and manage the application:

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

**Step 8: Learn and Explore Further**

1. **Spring Data** for database integration.
2. **Spring Security** for authentication and authorization.
3. **Spring Cloud** for microservices.

**Conclusion**

You’ve successfully learned the basics of Spring Core and Spring Boot. Experiment with creating additional REST APIs, connecting to databases, and exploring Spring's advanced features to deepen your understanding.