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

1. **Download Dependencies:**  
   
2. **Create Package Structure:**
   * com.example.di.demo
   * com.example.di.demo.service
   * com.example.di.demo.component
   * com.example.di.demo.config

#### STEP 2: Define Components and Services

Let's create some simple classes to demonstrate DI.

* **MessageSender.java** (Interface in com.example.di.demo.component)  
  // src/main/java/com/example/di/demo/component/MessageSender.java  
  package com.example.di.component;  
    
  public interface MessageSender {  
   void sendMessage(String message);  
  }
* **EmailSender.java** (Annotation-based component in com.example.di.demo.component)  
  // src/main/java/com/example/di/demo/component/EmailSender.java  
  package com.example.di.demo.component;  
    
  import org.springframework.stereotype.Component;  
    
  @Component("emailSender") // Marks as Spring component with a specific name  
  public class EmailSender implements MessageSender {  
   @Override  
   public void sendMessage(String message) {  
   System.out.println("Email Sent: " + message);  
   }  
  }
* **SmsSender.java** (Annotation-based component in com.example.di.demo.component)  
  // src/main/java/com/example/di/demo/component/SmsSender.java  
  package com.example.di.demo.component;  
    
  import org.springframework.stereotype.Component;  
    
  @Component("smsSender")  
  public class SmsSender implements MessageSender {  
   @Override  
   public void sendMessage(String message) {  
   System.out.println("SMS Sent: " + message);  
   }  
  }
* **NotificationService.java** (Service with DI in com.example.di.demo.service)  
  // src/main/java/com/example/di/demo/service/NotificationService.java  
  package com.example.di.demo.service;  
    
  import com.example.di.demo.component.MessageSender;  
  import org.springframework.beans.factory.annotation.Autowired;  
  import org.springframework.beans.factory.annotation.Qualifier;  
  import org.springframework.stereotype.Service;  
    
  @Service // Marks as Spring service component  
  public class NotificationService {  
    
   private MessageSender sender; // Dependency  
    
   // --- Constructor Injection (Recommended for mandatory dependencies) ---  
   @Autowired // Tells Spring to inject dependencies via this constructor  
   // @Qualifier("emailSender") Specifies which MessageSender bean to inject  
   public NotificationService(@Qualifier("emailSender") MessageSender sender) {  
   this.sender = sender;  
   System.out.println("NotificationService created via Constructor Injection with " + sender.getClass().getSimpleName());  
   }  
    
   // --- Setter Injection (Optional dependencies, or when changing at runtime) ---  
   // Uncomment to try Setter Injection instead of or in addition to constructor injection  
   /\*  
   private MessageSender secondarySender;  
    
   @Autowired  
   @Qualifier("smsSender")  
   public void setSecondarySender(MessageSender secondarySender) {  
   this.secondarySender = secondarySender;  
   System.out.println("Secondary sender set via Setter Injection with " + secondarySender.getClass().getSimpleName());  
   }  
   \*/  
    
   // --- Field Injection (Least recommended, but common for quick demos) ---  
   // Uncomment to try Field Injection (and comment out constructor/setter @Autowired)  
   /\*  
   @Autowired  
   @Qualifier("emailSender")  
   private MessageSender fieldInjectedSender;  
   \*/  
    
   public void sendNotification(String message) {  
   sender.sendMessage(message);  
   // if (secondarySender != null) secondarySender.sendMessage("Secondary: " + message);  
   // if (fieldInjectedSender != null) fieldInjectedSender.sendMessage("Field Injected: " + message);  
   }  
  }

#### STEP 3: Spring Configuration (Annotations & Java Config)

We'll use @ComponentScan for annotation-based wiring and @Bean methods for Java Config wiring.

* **AppConfig.java** (in com.example.di.demo.config)  
  // src/main/java/com/example/di/demo/config/AppConfig.java

package com.example.di.demo.config;

import com.example.di.demo.component.MessageSender;

import com.example.di.demo.component.SmsSender; // Import SmsSender for Java Config bean

import com.example.di.demo.service.NotificationService; // Import NotificationService for Java Config bean

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope; // For bean scopes

import org.springframework.beans.factory.annotation.Qualifier; // Import Qualifier

@Configuration // Marks this class as a Spring configuration source

@ComponentScan(basePackages = "com.example.di.demo") // Scans for @Component, @Service, etc.

public class AppConfig {

// --- Java Config Method for Wiring Beans ---

// This method explicitly defines a bean named "javaConfigSmsSender"

// Spring will call this method to get an instance of SmsSender.

@Bean("javaConfigSmsSender")

public MessageSender smsSenderViaJavaConfig() {

System.out.println("Creating SmsSender via Java Config @Bean method.");

return new SmsSender();

}

// This method defines a NotificationService bean that uses the smsSenderViaJavaConfig.

// Spring automatically resolves the dependency 'sender' by looking for a matching bean.

@Bean("javaConfigNotificationService")

// @Scope("prototype") // Uncomment to demonstrate prototype scope for this specific bean

public NotificationService notificationServiceViaJavaConfig(@Qualifier("javaConfigSmsSender") MessageSender sender) { // Added @Qualifier

System.out.println("Creating NotificationService via Java Config @Bean method.");

// We are explicitly passing the 'javaConfigSmsSender' bean here.

return new NotificationService(sender);

}

// --- Demonstrating Bean Scopes with @Bean ---

// By default, @Bean methods create singleton beans.

// To make it prototype, use @Scope("prototype")

@Bean("prototypeSmsSender")

@Scope("prototype") // Each request for this bean will return a new instance

public MessageSender prototypeSmsSender() {

System.out.println("Creating a NEW prototype SmsSender.");

return new SmsSender();

}

}

#### STEP 4: Main Application Class

This class will start the Spring IoC container and demonstrate retrieving beans wired using both annotations and Java Config, and observe bean scopes.

* **DiDemoApplication.java** (in com.example.di.demo)  
  // src/main/java/com/example/di/demo/DiDemoApplication.java

package com.example.di.demo;

import com.example.di.demo.component.MessageSender;

import com.example.di.demo.config.AppConfig;

import com.example.di.demo.service.NotificationService;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class DiDemoApplication {

public static void main(String[] args) {

System.out.println("--- Starting Spring IoC Container ---");

// Initialize Spring container with Java-based configuration

try (AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class)) {

System.out.println("\n--- Demonstrating Annotation-based Wiring (Autowired & ComponentScan) ---");

// Retrieve the NotificationService bean.

// Spring finds it via @ComponentScan and injects EmailSender via @Autowired @Qualifier("emailSender").

// Explicitly request by bean name to avoid NoUniqueBeanDefinitionException

NotificationService annotationNotificationService = (NotificationService) context.getBean("notificationService"); // Changed to retrieve by name

annotationNotificationService.sendNotification("Hello from Annotation-wired service!");

System.out.println("\n--- Demonstrating Java Config Wiring (@Bean methods) ---");

// Retrieve the NotificationService bean wired via Java Config.

// It's explicitly named "javaConfigNotificationService" in AppConfig.

NotificationService javaConfigNotificationService = (NotificationService) context.getBean("javaConfigNotificationService");

javaConfigNotificationService.sendNotification("Hello from Java Config-wired service!");

System.out.println("\n--- Demonstrating Bean Scopes ---");

// --- Singleton Scope (Default for @Component/@Service and @Bean) ---

// Retrieve the EmailSender bean. It's a singleton by default.

MessageSender emailSender1 = (MessageSender) context.getBean("emailSender");

MessageSender emailSender2 = (MessageSender) context.getBean("emailSender");

System.out.println("EmailSender instances are the same (Singleton)? " + (emailSender1 == emailSender2)); // Should be true

// --- Prototype Scope (Explicitly defined with @Scope("prototype")) ---

// Retrieve the prototypeSmsSender bean. Each call gets a new instance.

MessageSender prototypeSmsSender1 = (MessageSender) context.getBean("prototypeSmsSender");

MessageSender prototypeSmsSender2 = (MessageSender) context.getBean("prototypeSmsSender");

System.out.println("PrototypeSmsSender instances are the same (Prototype)? " + (prototypeSmsSender1 == prototypeSmsSender2)); // Should be false

}

System.out.println("\n--- Spring IoC Container Shut Down ---");

System.out.println("Activity Complete! You've explored DI basics, wiring methods, and bean scopes.");

}

}

### How to Run the Application:

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

You will see console output demonstrating the creation of beans, the injection of dependencies using both annotation-based and Java-based configuration, and the difference in behavior between singleton and prototype scoped beans.