Spring Boot Dependency Injection - Detailed Exercises

Exercise 1: Basic Constructor-Based Injection

Create a simple service-oriented structure using interfaces and classes:

1. Define an interface MessageService with a method getMessage().

2. Create HelloMessageService that implements MessageService and returns a static greeting.

3. Create a class MessagePrinter that takes a MessageService in its constructor and prints the message.

4. Annotate HelloMessageService and MessagePrinter with @Component and use Spring's ApplicationContext in the main method to get the MessagePrinter bean and call printMessage().

Goal: Understand how Spring auto-wires dependencies using constructor injection.

Exercise 2: Injecting Multiple Dependencies

Create a service class that requires two separate services to function:

1. Define NotificationService and LoggerService interfaces and create basic implementations.

2. Create UserService class that accepts both NotificationService and LoggerService via constructor.

3. Use @Component annotations and let Spring inject them automatically.

4. In UserService, create a createUser() method that uses both services to simulate real behavior.

Goal: Learn how to inject multiple dependencies into one component.

Exercise 3: Multiple Implementations with @Qualifier

Explore how Spring handles multiple implementations of the same interface:

1. Define Transport interface. Create Car and Bike implementations.

2. Create Traveller class that requires a Transport object.

3. Use @Qualifier in the constructor to specify which implementation to inject.

4. Use ApplicationContext to retrieve the Traveller bean and demonstrate usage.

Goal: Learn to control which implementation is injected when multiple beans match.

Exercise 4: Compare Field, Setter, and Constructor Injection

Compare three types of injection styles Spring supports:

1. Create SystemChecker which needs CpuChecker, MemoryChecker, and DiskChecker.

2. Inject dependencies using fields in one version, setters in another, and constructor in a third.

3. Log each injection and explain the advantages and downsides.

Goal: Understand the different types of dependency injection and why constructor injection is generally preferred.

Exercise 5: Manual Bean Configuration with @Bean

Practice configuring beans manually instead of using @Component:

1. Create Shape interface with Circle and Square implementations.
2. Do not annotate these with @Component.
3. Create a Configuration class (ShapeConfig) where you define @Bean methods to return Circle or Square.
4. Inject the Shape bean into a ShapeManager class.

Goal: Learn how to manually register beans using @Bean and @Configuration.

Exercise 6: Injection via Configuration Instead of Qualifiers

Use configuration classes to choose an implementation without using @Qualifier:

1. Define Color interface with Red and Green implementations.
2. Create Painter class that depends on Color.
3. Instead of using @Qualifier, create a Config class where you manually wire the desired implementation and pass it to Painter.

Goal: Understand how to control injection externally without polluting business logic with qualifiers.

Exercise 7: Nested Dependency Injection

Model a real-world scenario where one dependency itself has another dependency:

1. Create FuelPump class.
2. Create Engine class that needs FuelPump.
3. Create Car class that needs Engine.
4. Use constructor injection for all classes and print logs showing the injection chain.

Goal: Learn how nested dependencies are automatically resolved by Spring.

Exercise 8: Optional Dependency Injection

Spring Boot Dependency Injection - Detailed Exercises

Sometimes a dependency might not be present, and that should be okay:

1. Create LoggerService.

2. Create CacheService which optionally uses LoggerService (mark it with @Autowired(required = false)).

3. Test the app with and without LoggerService bean present.

Goal: Learn how Spring handles optional or missing beans gracefully.

Exercise 9: Same Bean Injected into Multiple Classes

Use one bean instance in multiple classes to understand singleton behavior:

1. Create RandomNumberGenerator class with a method to return random numbers.

2. Inject it into two different classes: Dice and LotteryMachine.

3. Retrieve and call both from main(), showing shared state or output if applicable.

Goal: Understand Spring's default singleton scope and shared instances.

Exercise 10: Interface Injection Controlled via Config Class

Handle multiple strategies through external configuration:

1. Create PaymentProcessor interface with PaypalProcessor and StripeProcessor implementations.

2. Create PaymentManager that uses a PaymentProcessor.

3. Use a Config class to choose which implementation to inject.

Goal: Practice interface-based design and switch strategies using Spring config.