# 1. Introduce Spring

## 1.1 About this book

## 1.2 What is Spring?

-Spring: a lightweight framework for building Java applications.

+You can use Spring to build any application in Java

+lightweight isn’t related to number of classes or the size of distribution, it’s the principle of Spring philosophy: minimal impact: you have to make few changes to app to gain benefits

### 1.2.1 Evolution of Spring Framework

(See in book)

### 1.2.2 Spring Projects

spring.io/projects

(See in book)

## 1.3 Inverting Control or Injecting Dependencies

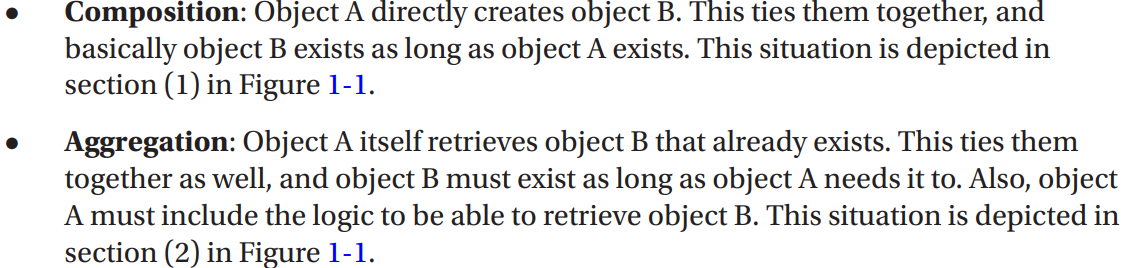
-The core of Spring Framework is based on the principle of **inversion of control (IoC)**: a technique that externalizes the creation and management of component dependencies. The action performed by any program is the result of interaction between its interdependent components.

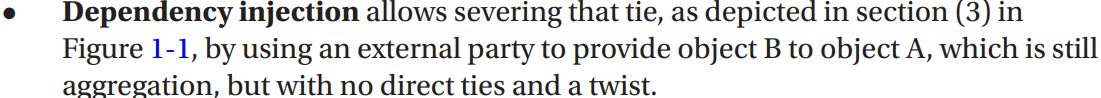
-**Dependency injection (DI)**: a concept that describes how dependent objects are connected at runtime by an external party.

-Object A needs an object of type B to perform its functions -> A depends on B. The 3 ways these 2 objects get to interact:

A screenshot of a computer

AI-generated content may be incorrect.





-**Inversion of control**: a design principle in which generic components are used to control the execution of problem-specific code, as in retrieving dependencies. Spring is a **dependency handler** used to perform DI, it was designed following IoC principle

-Spring’s DI implementation based on 2 core Java concepts: JavaBeans (POJOs-plan old Java objects) and interfaces.

-You can define dependency configuration in different ways: XML, annotation, Java configuration classes, Groovy-based configuration

-JavaBeans (POJOs) provides a standard mechanism for creating Java resources that are configurable in a number of was (constructors, setter methods). Any Spring-managed resource is bean.

-Interfaces and DI are technologies that are mutually beneficial. Designing and coding application to interfaces makes for a flexible application, but the complexity of wiring together is high. Using DI, you reduce the code you need to use an interface-based design in app.

+The use of interfaces allows dynamic proxies (Proxy pattern) to provide AOP.

-In context of DI, Spring acts like a container: provide instances of your application classes with all dependencies they need. Using Spring for DI follows JavaBeans naming conventions within your classes

### 1.3.1 Evolution of Dependency Injection

(See in books)

### 1.3.2 Beyond Dependency Injection

(See in books)

## 1.4 The Spring Community

# 2. Getting Started

## 2.1 Conventions

## 2.2 Who this book is for

## 2.3 What you need for this book

## 2.4 Prepare development environment

## 2.5 Understand Spring Packaging

-Spring packaging is modular: it allows to pick and choose which components you want to use in your application and to include only those components when you are distributing your app.

-Spring modules are JAR files that package the required code for that module.

-Spring Framework 6.0 comes with 22 modules. These name like: spring-<>-6.0.0.jar

A screenshot of a computer

AI-generated content may be incorrect.

A close up of a text

AI-generated content may be incorrect.

A close up of a text

AI-generated content may be incorrect.

A close up of a text

AI-generated content may be incorrect.

A close-up of a text

AI-generated content may be incorrect.

A close-up of a text

AI-generated content may be incorrect.

-If you sing Spring Boot, the appropriate set of Spring dependencies are configured depending on Spring Boot starter dependencies used. There are more than 30 of them

A close up of a text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## 2.6 Choose Modules for your application

-Without dependency management tool (Maven, Gradle), choosing which modules to use in application may be tricky.

### 2.6.1 Access Spring Modules on Maven Repository

### 2.6.2 Access Spring Modules Using Gradle

-Gradle is a powerful build tool use Groovy. Starting with version 4.x, Spring team has switched to using Gradle for configuration.

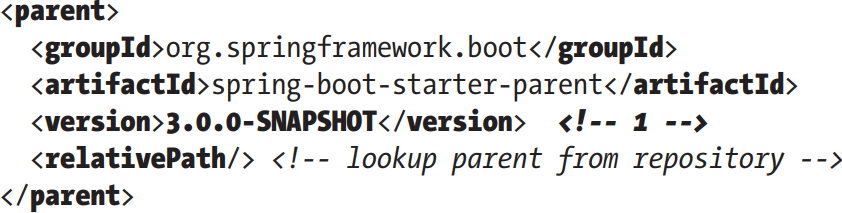
-The default name of Gradle configuration file for project is build.gradle.

A computer code with black text

AI-generated content may be incorrect.

### 2.6.3 Use Spring Boot Dependency Management

-You can use Spring Boot starter project as a parent project to provide your project with a minimal set of dependencies and default configuration.



-<dependencyManagement>

A screen shot of a computer program

AI-generated content may be incorrect.

-Gradle Spring Boot

A screen shot of a computer code

AI-generated content may be incorrect.

-In bigger projects, organized in multiple modules, configuration can create configuration templates that can be reused in modules.

## 2.7 Use Spring Documentation

docs.spring.io/spring-framework/docs/current/javadoc-api/

docs.spring.io/spring-framework/reference/

## 2.8 Putting a Spring into Hello World

### 2.8.1 Build sample

-Refactor the rendering and message retrieval login into separate components. And we should have these components implement interfaces and define interdependencies between components and launcher using these interfaces.

-Refactor message retrieval logic

A close-up of a message

AI-generated content may be incorrect.

-Interface for render message

A close-up of a message

AI-generated content may be incorrect.

-MessageProvider is a dependency of MessageRenderer

-Implementations

A computer screen shot of a computer code

AI-generated content may be incorrect.

A computer screen shot of a message

AI-generated content may be incorrect.

A screen shot of a computer program

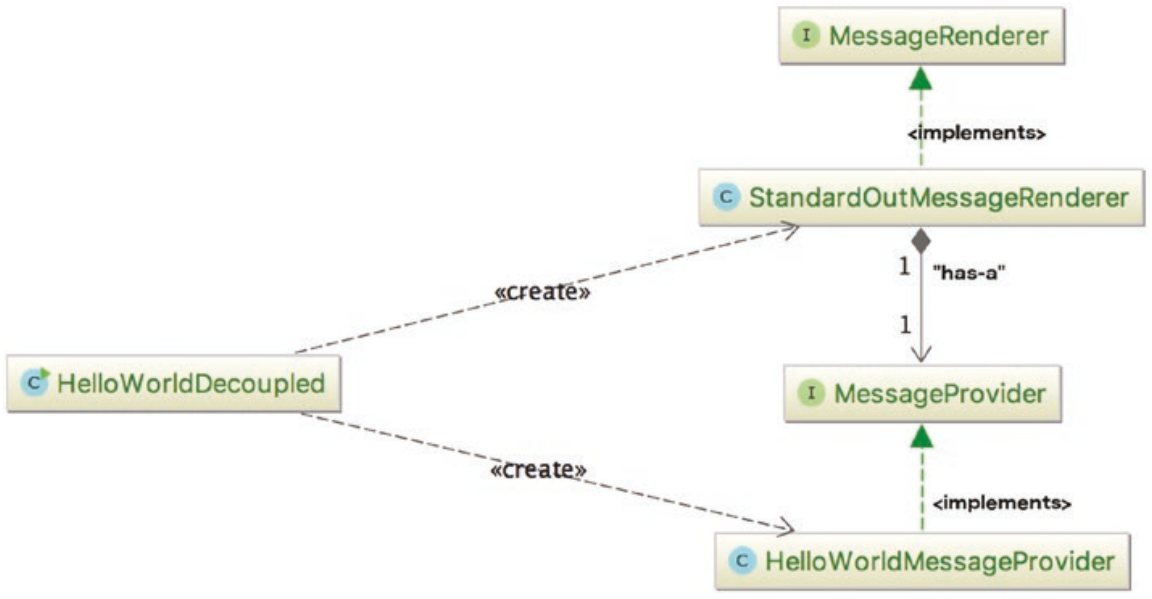
AI-generated content may be incorrect.

-Rewrite the main()

A screen shot of a computer code

AI-generated content may be incorrect.

-Abstract schema:



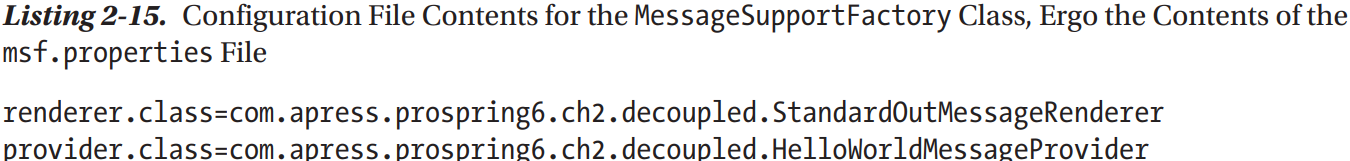
-Changing the implementation of interfaces means a change to code. The manual one is to create a simple factory class reading implementations class name from a property file and instantiates them.

A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.



A computer screen shot of text

AI-generated content may be incorrect.

-There is already a class **ServiceLoader** that does the exact thing like MessageSupportFactory. This class facilitates discovering and loading implementations matching a given interface. The interfaces that ServiceLoader retrieves implementations for are called **Service Provider Interface** (SPI)

-3 rules for configuration:

A close up of a name

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

### 2.8.2 Refactor with Spring

-Use Spring XML Configuration:

+Spring interface: ApplicationContext for storing all environmental information. This interface extends ListableBeanFactory: acts as the provider for any bean instance.

A screen shot of a computer code

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

+main() just obtain MessageRenderer bean by getBean(). Spring has created MessageProvider instance and injected it into MessageRenderer instance. We didn’t have to make changes to classes that are being wired together using Spring.

-Spring Configuration Using Annotation

+Since Spring 3.0, XML configuration files can be replaced with annotations and Java configuration classes

+**Configuration classes**: Java classes annotated with **@Configuration** that contain **bean definitions** (methods annotated with **@Bean**) or are configured themselves to identify bean definitions by annotating them with **@ComponentScanning**.

A computer screen shot of text

AI-generated content may be incorrect.

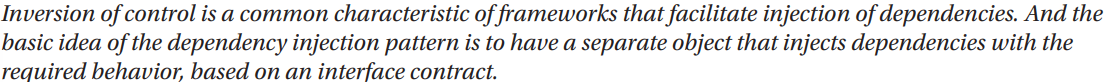
A screen shot of a computer code

AI-generated content may be incorrect.

+**AnnotationConfigApplicationContext**: ApplicationContext implementation read bean definitions from configuration classes

# 3. Introduce IoC and DI in Spring

-IoC and DI from Martin Flower:



## 3.1 Inversion of Control and Dependency Injection

-IoC aims to offer a simpler mechanism for provisioning component dependencies (object’s collaborators) and managing these dependencies throughout their life cycle.

-A component that requires certain dependencies is dependency object/target.

-In general, IoC can be decomposed into 2 subtypes: dependency injection and dependency lookup. These subtypes are decomposed into concrete implementations of IoC services.

### 3.1.1 Types of IoC

-IoC is more of a mixture of old and new ideas. Dependency lookup: a component must acquire a reference to a dependency. DI: the dependencies are injected into component by IoC container.

-Dependency lookup comes in 2 types:

A close up of a text

AI-generated content may be incorrect.

-Dependency injection has 2 types:

A close up of words

AI-generated content may be incorrect.

-Dependency Pull

+The most familiar type of IoC.

+Dependencies are pulled from a registry as required. Anyone who has written code to access an EJB has used dependency pull (via JNDI API to look up EJB component)

A diagram of a system

AI-generated content may be incorrect.

+Spring offers dependency pull as a mechanism for retrieving components that the framework manages In chap 2, we used it:

A screenshot of a computer code

AI-generated content may be incorrect.

+MessageRenderer bean is retrieved from ApplicationContext that functions as a register of all beans in application. This instance is pulled, so render() can be invoked.

-Contextualized Dependency Lookup:

+Lookup is performed against the container that is managing the resource, it usually performed at some set point.

A diagram of a process

AI-generated content may be incorrect.

+CDL works by having the components that requires a dependency implement an interface. A component is use this interface to signal to container that it wants to obtain a dependency.

A close-up of a computer screen

AI-generated content may be incorrect.

+Container is usually provided by application server(Tomcat, JBoss) or framework (Spring).

A white background with black text

AI-generated content may be incorrect.

+When container is ready to pass dependencies to component, it calls performLookup() on each component in turn. Component can then look up its dependencies by using Container interface

A screenshot of a computer program

AI-generated content may be incorrect.

-Constructor DI

+IoC container injects component’s dependencies via its constructor. Component declares constructor(s), taking arguments as dependencies, IoC container passes dependencies to component when instantiation occurs.

A screenshot of a computer program

AI-generated content may be incorrect.

+Consequence: object can’t created without its dependencies.

### 3.1.2 Setter DI

-IoC container injects dependencies via JavaBean-style setter.

A screenshot of a computer program

AI-generated content may be incorrect.

+Object can be crated without its dependencies, they can be provided later.

-Injection vs Lookup : (See in book)

### 3.1.3 Setter Injection vs Constructor Injection

-Setter Injection: allows dependencies swapped out without creating new objects, let your class choose appropriate default.

-Constructor Injection: ensure dependencies are being passed to component, when designing for immutable objects.

## 3.2 IoC in Spring

-IoC is a big part of what Spring does. The core of Spring’s implementation is based on DI.

A close-up of a sign

AI-generated content may be incorrect.

-Although DI is preferred mechanism for wiring together collaborators and dependent objects, you need dependency lookup to access dependent objects.

-Spring’s IoC container has the ability to act as an adapter between own DI container and external dependency lookup containers

## 3.3 DI in Spring

### 3.3.1 Beans and BeanFactory

-**org.springframework.beans** and **org.springframework.context** packages are the basic for Spring Framework’s IoC container.

-The central point of Spring’s IoC container is **org.springframework.beans.factory.BeanFactory** interface. Implementations of this interface are responsible for managing components: dependencies and their life cycles

A diagram of a factory

AI-generated content may be incorrect.

-**bean**: any object managed by Spring IoC container. Container creates, configures (assembles) and manages beans throughout their life cycle.

-If app needs only DI support, you can interact with container via BeanFactory.

-ApplicationContext is an extension of BeanFactory

A close up of text

AI-generated content may be incorrect.

+It’s recommended that you interact with Spring via ApplicationContext. Spring support manual coding (instantiate it manually and load configuration) or in a web container environment via ContextLoaderListener

### 3.3.2 Configuring ApplicationContext

-For simple, Spring apps are configured through Java annotations and Java code

### 3.3.3 Basic Configuration Overview

-@Configuration: The class contains methods @Bean (bean declarations). This approach works for any types of object, especially 3rd party libraries

-@ComponentScan: enable class for looking existing bean declaration. The discoverable bean declarations are classes @Component and other stereotype annotations.

-All these annotations describe **configuration metadata**

A computer screen shot of a program

AI-generated content may be incorrect.

-**Configuration classes** are **bootstrapped** using AnnotationConfigApplicationContext or AnnotationConfigWebApplicationContext

A computer screen shot of a program

AI-generated content may be incorrect.

+By instantiating this class, we created an instance of Spring IoC container that read bean declaration, create the beans, add them to its registry, and manage them. Use a reference to container, beans can be retrieved and used.

### 3.3.4 Declare Spring Components

-**Stereotype** annotations: another way to declare bean

+Define the role of types or methods.

+org.springframework.stereotype

A close up of a text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

-@ComponentScan: Spring will seek out component classes, instantiate beans

A black text on a white background

AI-generated content may be incorrect.

+By default, scanning occurs from configuration class package to subpackages.

+basePackages: declare package or a collection of packages for looking components

A computer code with black text

AI-generated content may be incorrect.

+It’s a time-consuming operation, good practice is try to limit the places Spring will look for bean definition

A white background with black text

AI-generated content may be incorrect.

**-@ImportResource**: import bean definitions from XML file (or more)

**-@Import**: Import other bean definitions from Java configuration classes

### 3.3.5 Use Seter Injection

-To configure setter injection, @Autowired must be put on every setter to inject a dependency.

A screen shot of a computer program

AI-generated content may be incorrect.

-Instead of @Autowired, you can use @Resource(name=”provider”) . It’s one of annotations in JSR-250. It supports name parameter for more fine-grained DI requirements

-@Inject from JSR-330 = @Autowired

### 3.3.6 Use Constructor Injection

-Make sure a bean is not even created without its dependencies, you can declare dependency as argument for constructor

A screen shot of a computer program

AI-generated content may be incorrect.

-In Spring 4.X, Spring IoC call only constructor present to create the bean regardless of presence/absence of annotation. Single constructor initializing all dependencies doesn’t require @Autowiring too.

### 3.3.7 Use Field Injection

-This is the 3rd type of DI supported in Spring

-Dependency is injected into filed.

A screen shot of a computer code

AI-generated content may be incorrect.

+The field is private, but container uses reflection to populate the dependency

A white text with black text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

-Field injection isn’t recommended (can’t used for final fields, difficulties writing tests…). Howerver, it’s practical to use it for @Configuration and @Test clases (Integration Test)

## 3.4 Use Injection Parameters

-Spring support many options for injection parameters=inject simple values and Java collections, externally defined properties, beans in another factory

### 3.4.1 Inject Simple Values

-@Value: inject simple values into beans. By default, it can read String, but can convert to primitive or primitive wrapper.

A screenshot of a computer code

AI-generated content may be incorrect.

### 3.4.2 Inject Values Using SpEL

-Spring Expression Language (SpEL): evaluate an expression dynamically and then use it in ApplicationContext. You can use the result for injection into beans.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

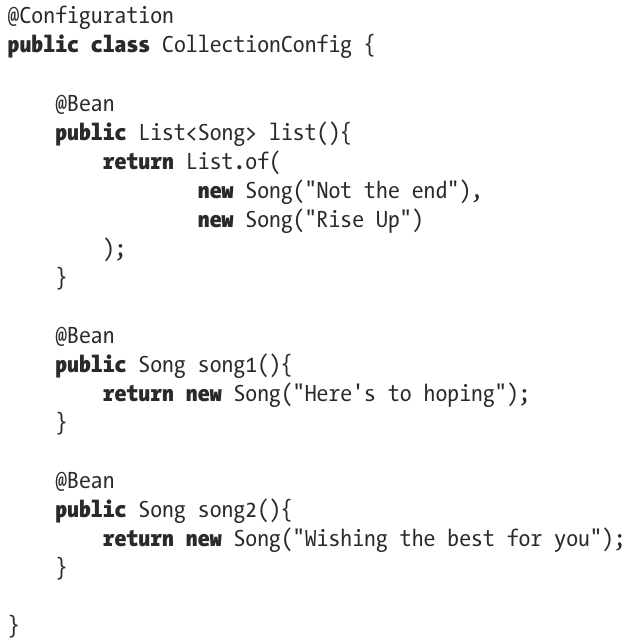
### 3.4.3 Injection and ApplicationContext Nesting

-Spring supports a hierarchical structure for ApplicationContext so that one context is considered the parent of another. It allows you to split configuration into different files

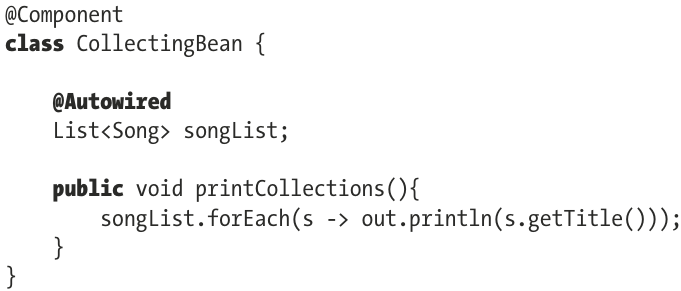
(See in books)

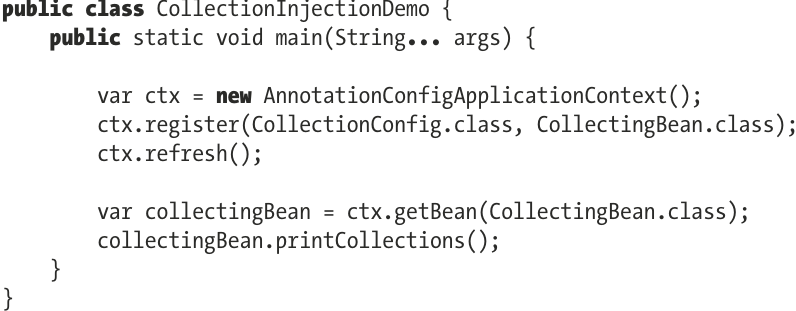
### 3.4.4 Inject Collections

-Often beans need access to collections of objects rather than individual beans or values. Just declare method @Bean in configuration class that returns List, Set, Properties, Map.



-Inject list bean:





+@Autowired treats arrays, collections and maps as sets of corresponding beans, with target bean type derived from declared collection value type. So with List<Song>, Spring will try to inject all beans of type Song in current ApplicationContext.

-For collection type injection, we have to explicitly instruct Spring to perform injection by @Qualifier in springframework.beans.factory.annotation.

-Beside, there are 3 additional ways:

+@Inject @Named(“list”) in jakarta.inject: @Inject=@Autowired and @Named=@Qualifier

+@Resource(name=”list”) in jakarta.annotation

[+@Value(“#{colelctionConfig.list}”)](mailto:+@Value(“#{colelctionConfig.list}”))” the recommended way to inject collections

-**Note**: The behavior above can applies for Set, Mapp: Map will inject {beanName, bean}

## 3.5 Use Method Injection

-Another less frequently used DI feature that Spring provides. It comes in 2 loosely related forms:

+Lookup Method Injection: another mechanism by which a bean can obtain one of dependencies

+Method Replacement: replace implementation of any method on bean arbitrarily

-(See in book)

### 3.5.1 Lookup Method Injection

### 3.5.2 Considerations for Lookup Method Injection

## 3.6 Understand Bean Naming

### 3.6.1 Default Bean Naming Style for Beans Declared with @Component

### 3.6.2 Customize Bean Naming Style

### 3.6.3 Bean Naming Style for Beans Declared with @Bean

### 3.6.4 Explicit Bean Naming

### 3.6.5 The @AliasFor Annotation

### 3.6.6 Understand Bean Instantiation Mode

## 3.7 Choose an Instantiation Mode

## 3.8 Additional Bean Scope

## 3.9 Resolving Dependencies

## 3.10 Autowiring Your Bean

### 3.10.1 Constructor Autowiring

### 3.10.2 byType Autowiring

### 3.10.3 byName Autowiring

### 3.10.4 Yet Another Pickle

### 3.10.5 When to Use Autowiring

# 4. Advanced Spring Configuration and Spring Boot

## 4.1 Spring’s Impact on Application Portability

### 4.2 Bean Life-Cycle Management

## 4.3 Hooking into Bean Creation

### 4.3.1 Execute a Method when a Bean is created

### 4.3.2 Implement the InitializingBean Interface

### 4.3.3 Use the JSR-250 @PostConstruct

### 4.3.4 Understand Order of Resolution

## 4.4 Hooking into Bean Destruction

## 4.5 Execute a Method when a Bean is Destroyed

### 4.5.1 Implement the DisposableBean Interface

### 4.5.2 Use JSR-250 @PreDestroy

### 4.5.3 Understand Order of Resolution

### 4.5.4 Use a Shutdown Hook

## 4.6 Make your Beans “Spring Aware”

### 4.6.1 Use BeanNameAware Interface

### 4.6.2 Use ApplicationContextAware Interface

## 4.7 Use of FactoryBeans

### 4.7.1 FactoryBean Example

### 4.7.2 Access a FactoryBean directly

## 4.8 JavaBeans PropertyEditors

### 4.8.1 Use the Built-in PropertyEditors

### 4.8.2 Create a custom PropertyEditor

## 4.9 More Spring ApplicationContext Configuration

## 4.10 Internationalization

### 4.10.1 Internationalization with MessageSource

### 4.10.2 Use getMessage()

### 4.10.3 Why use ApplicationContext as a MessageSource?

### 4.10.4 Use MessageSource in Stand-Alone Applications

### 4.10.5 Events Publication

### 4.10.6 Use Application Events

### 4.10.7 Considerations for Event Usage

## 4.11 Access Resources

## 4.12 Advanced Java Configuration Classes

## 4.13 Profiles

### 4.13.1 Example

### 4.13.2 Considerations for Using Profiles

## 4.14 Environment and PropertySource Abstraction

## 4.15 Testing Spring Applications

### 4.15.1 Use Spring Test Annotations

### 4.15.2 Implement Logic Unit Tests

### 4.15.3 Implement an Integration Test

## 4.16 Configure Profile for Integration Testing

### 4.16.1 Implement a Front-End Unit Test

### 4.16.2 Introduce Selenium

## 4.17 Configuration Using Groovy

## 4.18 Use Spring Boot