***Spring***

***Framework***

***Notes***

Table of Contents

[What is the Spring Framework? 5](#_Toc166251915)

[Key Features of Spring 5](#_Toc166251916)

[Dependency Injection (DI) and Inversion of Control (IoC) 5](#_Toc166251917)

[Modules 5](#_Toc166251918)

[Data Access/Integration 5](#_Toc166251919)

[Web Module 6](#_Toc166251920)

[AOP (Aspect-Oriented Programming) 6](#_Toc166251921)

[Instrumentation and Tester Modules 6](#_Toc166251922)

[Practical Applications of Spring 6](#_Toc166251923)

[Starting with Spring 6](#_Toc166251924)

[Spring Inversion of Control (IoC) Container 7](#_Toc166251925)

[IoC Container Overview 7](#_Toc166251926)

[Description of the Inversion of Control Container 7](#_Toc166251927)

[Core Functions of the Inversion of Control Container 7](#_Toc166251928)

[Managing Object Creation and Lifecycle 7](#_Toc166251929)

[Dependency Injection 7](#_Toc166251930)

[Configuring Beans 7](#_Toc166251931)

[Managing Bean Lifecycles 8](#_Toc166251932)

[Inversion of Control Configuration Styles 8](#_Toc166251933)

[XML Configuration 8](#_Toc166251934)

[Annotation-Based Configuration 8](#_Toc166251935)

[Java-Based Configuration 8](#_Toc166251936)

[Benefits of Using an Inversion of Control Container 8](#_Toc166251937)

[Spring Modules Explained 8](#_Toc166251938)

[Background Description 9](#_Toc166251939)

[Overview of Spring Modules 9](#_Toc166251940)

[Core Container 9](#_Toc166251941)

[AOP (Aspect Oriented Programming) and Instrumentation 9](#_Toc166251942)

[Data Access/Integration 10](#_Toc166251943)

[Web and Servlet 10](#_Toc166251944)

[Messaging 11](#_Toc166251945)

[Testing 11](#_Toc166251946)

[\*Spring Module-View-Control (MVC) Module 11](#_Toc166251947)

[Basic Description 11](#_Toc166251948)

[Components of a Spring MVC 11](#_Toc166251949)

[Model 12](#_Toc166251950)

[View 12](#_Toc166251951)

[Controller 12](#_Toc166251952)

[How Spring MVC Works 12](#_Toc166251953)

[DispatcherServlet 12](#_Toc166251954)

[Handlers and Handler Mapping 12](#_Toc166251955)

[ViewResolver 12](#_Toc166251956)

[ModelAndView 13](#_Toc166251957)

[Workflow of a Request in Spring MVC 13](#_Toc166251958)

[Benefits of Using Spring MVC 13](#_Toc166251959)

[Basic Learning Concepts 14](#_Toc166251960)

[Dependency Injection 14](#_Toc166251961)

[Definition and Basic Concept 14](#_Toc166251962)

[How Dependency Injection Works (With Example) 14](#_Toc166251963)

[Types of Dependency Injection 15](#_Toc166251964)

[Benefits of Dependency Injection 16](#_Toc166251965)

[Dependency Injection in Frameworks 16](#_Toc166251966)

[Important Note about DI in Our Spring Work 16](#_Toc166251967)

[Beans 16](#_Toc166251968)

[Description 16](#_Toc166251969)

[Definition and Management 16](#_Toc166251970)

[Bean Scope 17](#_Toc166251971)

[Bean Lifecycle 17](#_Toc166251972)

[Bean Dependency Injection 17](#_Toc166251973)

[Example of a Bean Definition 17](#_Toc166251974)

[Best Practices and Bean Usage 18](#_Toc166251975)

[Auto-Wiring 18](#_Toc166251976)

[Definition and Basic Concept 18](#_Toc166251977)

[\*\*Basic Concept of Auto-wiring 19](#_Toc166251978)

[How Auto-Wiring Works (With Example) 19](#_Toc166251979)

[Configuring Autowiring in Java 19](#_Toc166251980)

[Benefits of Autowiring 20](#_Toc166251981)

[Considerations 20](#_Toc166251982)

[Important Next Steps 20](#_Toc166251983)

Spring Framework Reference Documentation

***Official Source (Spring Website):*** <https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/>

# What is the Spring Framework?

The Spring Framework is a powerful tool for building Java applications. It helps manage your code and makes it easier to develop robust applications quickly. The framework handles much of the boilerplate code for you, allowing you to focus on the unique features of your application.

## Key Features of Spring

### Dependency Injection (DI) and Inversion of Control (IoC)

* These are design principles that help manage dependencies between objects, making your code more modular and easier to manage. Instead of manually creating objects, Spring can inject these dependencies for you when they're needed.

### Modules

* Spring is made up of several modules that you can pick and choose to use depending on your application needs. These include data access, web application development, messaging, and transaction management, among others.

### Data Access/Integration

* Spring provides support for working with data through JDBC, ORM (like Hibernate), OXM, JMS, and transaction management modules. These tools help you interact with databases and manage data efficiently.

### Web Module

* This includes Spring MVC, which is a web framework that provides a clear separation between domain model code and web forms. It helps you develop web applications with a clean architecture.

### AOP (Aspect-Oriented Programming)

* AOP helps in separating cross-cutting concerns like transaction management or logging. It allows for cleaner modularization of functionalities that cut across multiple types and objects.

### Instrumentation and Tester Modules

* These provide support for class instrumentation and classloader implementations that are helpful in certain application servers. The tester module supports the testing of Spring components with JUnit or TestNG

## Practical Applications of Spring

* You can use Spring to ensure that methods in your application can perform functions like running within a database transaction, or being exposed as a web service, without having to write specialized code directly.
* Spring's declarative transaction management allows your application to rollback transactions if exceptions occur, without manual handling.

## Starting with Spring

**For beginners:**

* Focus on understanding how dependency injection works and how it can simplify the way you build applications.
* Explore Spring Boot, a part of Spring that offers "convention over configuration" to help you get a project up and running quickly.
* Consider learning through practice, perhaps starting with building a simple web application using Spring MVC.

# Spring Inversion of Control (IoC) Container

## IoC Container Overview

* In the Spring Framework, the Inversion of Control (IoC) container is a central feature that manages the creation, configuration, and lifecycle of application objects. This container is often referred to simply as the Spring context and is responsible for instantiating, configuring, and assembling objects known as beans.

## Description of the Inversion of Control Container

* The IoC container is the mechanism that implements inversion of control. It manages the construction and wiring of your components so that you don’t have to manually create objects and manage their dependencies. Instead, the container injects these dependencies when needed. This process is controlled through configuration metadata that you provide, which can be in the form of XML, annotations, or Java-based configuration.

## Core Functions of the Inversion of Control Container

### Managing Object Creation and Lifecycle

* The container creates and manages the lifecycle of beans defined in your Spring application.
  + It controls everything from creating object instances to destroying them when no longer needed.

### Dependency Injection

* The IoC Container injects dependencies into objects as defined in the configuration metadata.
  + This helps in managing the relationships between objects and promotes loose coupling.

### Configuring Beans

* Beans can be configured in a variety of ways through the container.
  + For instance, properties can be set, methods can be called, o even other beans can be constructed and set.

### Managing Bean Lifecycles

* The container also managed the lifecycle of beans, including calling initialization and destruction methods.
  + This allows for performing specific actions upon bean initialization and before bean destruction.

## Inversion of Control Configuration Styles

### XML Configuration

* Traditionally, beans were configured in XML files where each bean is defined and its dependencies are injected via XML tags.

### Annotation-Based Configuration

* Modern Spring applications often use annotations such as ‘@Autowired’, ‘@Service’, ‘@Repository’, and ‘@Component’, which minimize the need for XML configuration and make the codebase more readable.

### Java-Based Configuration

* Spring allows configuration to be done entirely in Java with the use of configuration classes annotated with ‘@Configuration’ and beans defined by methods annotated with ‘@Bean’.

## Benefits of Using an Inversion of Control Container

* **Decoupling:** Decouples the execution of a task from implementation, increasing flexibility and reusability of code
* **Ease of Testing:** Makes unit and integration testing easier.
* **Centralized Configuration:** Changes in object creation and management can be managed centrally, reducing the impact on existing code.

# Spring Modules Explained

## Background Description

* Spring Framework is modular, consisting of a series of components that can be used independently or collectively. Each module serves a specific purpose and can be integrated into your application as needed.

## Overview of Spring Modules

### Core Container

* This is the foundation of the Spring framework, consisting of the ‘spring-core’, ‘spring-beans’, ‘spring-context’, and ‘spring-expression’ modules. It manages configuration and wiring of Spring beans, provides dependency injection features, and more.

#### Spring-Core and Spring-Beans

* Provide the fundamental parts of the framework, including Inversion of Control and dependency injection.

#### Spring-Context

* This module extends the core and beans capabilities for enterprise-specific functionality such as event propagation, resources loading, and transparent creation of contexts.

#### Spring-Expression

* Provides a powerful expression language for querying and manipulating an object graph at runtime.

### AOP (Aspect Oriented Programming) and Instrumentation

* Supports aspect-oriented programming, enabling you to define method interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
  + This module provides the basis for transaction management and allows you to declare method interceptors programmatically.

#### Spring-aop

* Allows for dynamic cross-cutting concerns, such as transaction management.

#### Spring-aspects

* Integrates with AspectJ, which is a powerful aspect-oriented extension to Java.

### Data Access/Integration

* Simplifies data access from the database, through JDBC or object-relational mapping, with support for transactions.

#### spring-jdbc

* Simplifies the use of JDBC and helps to avoid common errors.

#### spring-tx

* Provides powerful transaction management capabilities.

#### spring-orm

* Integrates with popular ORM frameworks such as Hibernate, JPA, and JDO.

#### spring-oxm

* Supports Object/XML mapping.

### Web and Servlet

* Frameworks such as Spring MVC (Model-View-Controller) and Spring WebFlux are provided under this category for creating web applications.

#### spring-web

* Provides basic web-oriented integration features, such as multipart file upload functionality and initialization of the IoC container.

#### spring-webmvc

* Also known as Spring MVC, offers features to build web applications using the MVC design pattern.

#### spring-webflux

* Supports building reactive applications on the Spring model.

### Messaging

* The module for application integration. It includes support for messaging systems and works with the messaging APIs provided.

#### spring-messaging

* Provides support for messaging architectures and protocols.

### Testing

* Facilitates the testing of Spring components with JUnit or TestNG.

#### spring-test

* Provides support for testing Spring components with configuration via Spring ApplicationContexts.

*These modules make Spring a comprehensive framework suitable for all sorts of enterprise applications, from simple web applications to complex, high-transaction enterprise applications.*

* **YOU CAN JUST USE WHAT YOU NEED**, or you can leverage the entire framework as per the requirements of your project. Each module in Spring is designed to work independently although integration between modules is frequently done and supported.

# \*Spring Module-View-Control (MVC) Module

## Basic Description

* Spring MVC is a module within the Spring Framework that implements the Model-View-Controller (MVC) design pattern for building web applications. It is one of the most popular frameworks for creating scalable and robust web applications.

## Components of a Spring MVC

### Model

* Represents the data and the business rules of the application. It manages the data, logic, and the rules of the application

### View

* Responsible for rendering the model data and generally refers to the UI logic of the application.
  + In Spring MVC, views can be created using various technologies like JSP, Thymeleaf, or even templating engines such as Freemarker.

### Controller

* Acts as an intermediary between the model and the view
  + Processes user input and performs interactions on the data model objects
* The controller receives the input, validates it, and then performs business operations that modify the state of the data model.

## How Spring MVC Works

### DispatcherServlet

* The central controller of the Spring MVC framwork which handles all the HTTP requests and responses
  + It reads the configuration (defined in the XML, annotations, or Java config) to map requests to the appropriate controllers.

### Handlers and Handler Mapping

* Controllers are mapped to URL requests through handlers
* **HandlerMapping**
  + Is responsible for interpreting the user request and finding the correct controller

### ViewResolver

* Once the controller performs the business logic, it returns a model and a view name
* **ViewResolver**
  + Takes this view name and translates it into a specific view (like a JSP) that should be rendered

### ModelAndView

* A container object that includes both the model data and the logical view name
  + The controller returns this object

## Workflow of a Request in Spring MVC

1. **HTTP Request**: User sends a request to the server, which is received by the Spring **DispatcherServlet**.
2. **DispatcherServlet**: Consults the **HandlerMapping** to call the appropriate controller.
3. **Controller**: Executes the necessary logic and returns a **ModelAndView** object (model data and view name) back to the **DispatcherServlet**.
4. **View Resolver**: The **DispatcherServlet** uses the **ViewResolver** to find the specific view page to render by using the view name.
5. **View**: The view uses the data prepared by the controller to generate a final page.
6. **HTTP Response**: The rendered view is returned to the client as an HTTP response.

## Benefits of Using Spring MVC

* Flexible Configuration
  + Spring MVC supports both annotations and XML for configuration making it flexible and adaptable to different needs.
* Integration
  + Easily integrates with other Spring functionalities like Spring security, Spring Data, etc.
* Separation of Concerns
  + Follows the MVC pattern, which helps in organizing code to make it easier to develop, test, and maintain
* Reusable Business Logic
  + Controllers in Spring MVC can be reused across different applications with minimal changes

# Basic Learning Concepts

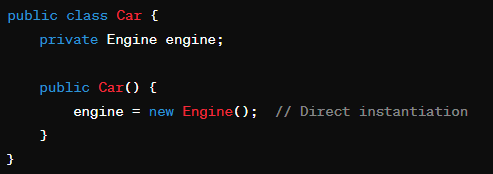
## Dependency Injection

### Definition and Basic Concept

* Dependency Injection (DI) is a design pattern used to implement inversion of control between classes and their dependencies. Through DI, objects receive their dependencies from an external source rather than creating them internally. This approach has several benefits, including reduced dependency coupling, improved testability, and greater separation of concerns.
* **Dependency Injection Involves 3 Core Elements:** 
  + **Dependency Injector:** The mechanism that injects dependencies into an object
    - \*\*Typically, a container in frameworks like ***Spring***
  + **Client:** The object that has dependencies
    - This could be a class that requires database access or network resources
  + **Service:** The dependency itself
    - Ex. A database connection pool or a data service

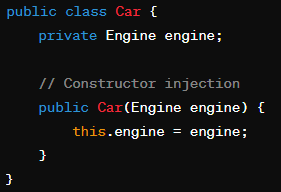
### How Dependency Injection Works (With Example)

If you have a class known as ‘Car’ that depends on an ‘Engine,’ without dependency injection you might typically instantiate an ‘Engine’ inside your ‘Car’ class similar to the code below…

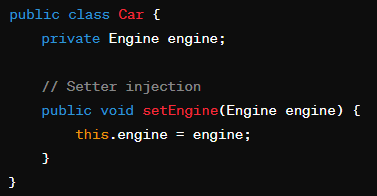
**

However, with dependency injection, you avoid hardcoding the dependency of the ‘Engine’ as it is instead passed into the ‘Car’ at runtime, possibly through an injector mechanism…

* Passing the dependency into the constructor is known as “Constructor Injection”



As an alternative, you can use “Setter Injection” where the dependency is instead set through a method…



### Types of Dependency Injection

#### Constructor Injection

* Dependencies are provided through the class constructor

#### Setter Injection

* Dependencies are set through public setters or other methods after the object is already constructed

#### Interface Injection (Less Common)

* The dependency provides an injector method that will inject the dependency into any client passed into it

### Benefits of Dependency Injection

* Decoupling Code: Components are less dependent on how their dependencies are created
* Ease of Testing: Easier to test components by mocking dependencies
* Flexibility: Components can be easily connected to different systems or configured for different settings
* Boilerplate Reduction: Centralized configuration reduces the need to repeatedly set up dependencies in each component

### Dependency Injection in Frameworks

* Frameworks like Spring use DI extensively to manage components. In Spring, objects managed by the Spring IoC (Inversion of Control) container can be easily wired together by merely configuring them in XML files or annotations

### Important Note about DI in Our Spring Work

* In our work, we will see Spring objects being wired through ‘Auto-wiring’, this will be explained further below

## Beans

### Description

* In the context of the Spring Framework, a "bean" is a fundamental concept. Beans are the objects that form the backbone of your application and are managed by the Spring IoC (Inversion of Control) container. Spring beans are instantiated, assembled, and otherwise managed by the Spring IoC container. These beans are created with the configuration metadata that you supply to the container, which could be in the form of XML configuration or annotations.

### Definition and Management

* Beans are typically defined in a Spring configuration file or through annotations and managed by the Spring container. The container handles the lifecycle of beans, including creating bean instances, wiring them together, configuring them, and managing their complete lifecycle.

### Bean Scope

* Beans can have different scopes, which define their lifecycle and visibility. The most common scopes are "singleton" (one instance per Spring IoC container) and "prototype" (a new instance every time the bean is requested).
  + This can be seen in the work of Students through Magenta Project 2, in which they ensured beans were only defined once in the program, otherwise leading to errors.

### Bean Lifecycle

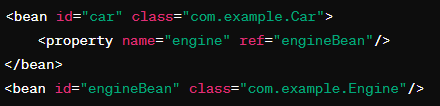
* The Spring container manages the complete lifecycle of a bean. It calls the lifecycle methods at the start and end of a bean’s life, providing hooks that you can use for initialization and destruction tasks.

### Bean Dependency Injection

* Beans have their dependencies injected by the spring container. This can be done via constructor injection, setter injection, or field injection, allowing for loose coupling and easy testing.

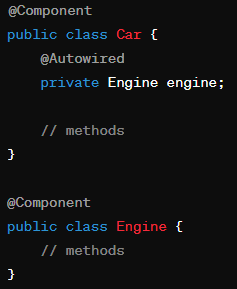
### Example of a Bean Definition

*Defining a Bean in an XML configuration File*



*In this example, ‘Car’ is a bean which depends on another bean ‘Engine’. The ‘<property>’ tag is used to inject the ‘Engine’ bean into the ‘Car’ bean…*

*Defining the same beans using annotations in java code:*



*In this case, the ‘@Component’ annotation tells Spring that ‘Car’ and ‘Engine’ are beans, and the ‘@Autowired’ annotation tells Spring to inject an ‘Engine’ instance into the ‘Car’…*

### Best Practices and Bean Usage

* Use appropriate scopes according to the bean usage
* Favor constructor injection as it promotes immutability and ensures that required dependencies are not null
* Utilize life cycle annotations like ‘PostConstruct’ and ‘PreDestroy’ for lifecycle management
* Keep bean classes clean and focussed on business logic, letting Spring handle infrastructure concerns

**BEANS ARE CRUCIAL AND CENTRAL TO HOW SPRING MANAGED THE APPLICATION CONTEXT AND DEPENDENCIES**

## Auto-Wiring

### Definition and Basic Concept

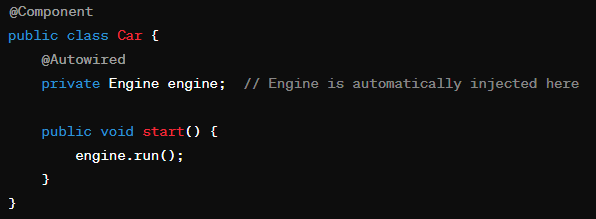
* Auto-wiring is a feature in Spring that allows you to automatically inject dependencies without explicitly defining them in your configuration files or annotations. This makes it easier to manage and wire components in large applications.

### \*\*Basic Concept of Auto-wiring

* Auto-wiring in Spring can automatically resolve the beans (dependencies) that your beans (components) need, by matching them in the Spring container (the context). This is done based on the type, name, or other qualifiers of the beans.

### How Auto-Wiring Works (With Example)

* There are many modes of Auto-Wiring in Spring
  + **No Auto-Wiring (Used by Default):** All dependencies must be injected manually in your configuration file or annotations
  + **byType:** Spring looks for a bean that matches the property’s type to inject
  + **byName:** Spring looks for a bean with the same name as the property to inject
  + **constructor:** Similar to byType, but applies to constructor arguments
  + **autodetect:** Spring first tries to auto-wire by constructor, then by type
* Below is an example using ‘@Autowired’, the annotation used in auto-wiring…



***^^ In this example, Spring looks for a bean of type ‘Engine’ and injects it into the ‘Car’ class where ‘Engine’ is annotated with ‘@Autowired’***

### Configuring Autowiring in Java

*Configuration for autowiring can both done in both XML configuration and annotations…*

#### XML Configuration

* ‘autowire’ attribute is used in the ‘<bean>’ tag
* 

#### Annotation Configuration

* Simply use ‘@Autowired’ on constructors, setters, or fields…

**A computer screen shot of text

Description automatically generated**

### Benefits of Autowiring

* Reduces boilerplate code needed for manual writing
* Increases the flexibility and reusability of components

### Considerations

* Autowiring can make it difficult to trace how dependencies are being injected, which can complicate debugging
* It requires that **EXACTLY ONE** bean of the property type exists in the container
  + If no beans or multiple candidate beans are found, it can lead to ERRORS

# Important Next Steps

* Advanced Notes on this links content (Spring Version 4.0.X) regarding specifics relating to the IoC container and Beans, including naming conventions and code examples…
  + <https://docs.spring.io/spring-framework/docs/4.1.9.RELEASE/spring-framework-reference/html/beans.html>