# **WEEK-3: Spring Core Maven , Spring Data JPA**

## **Introduction to Spring Framework**

**Overview of the Spring Framework**

The **Spring Framework** is a powerful, lightweight, and modular framework for building enterprise Java applications. It provides comprehensive infrastructure support for developing Java applications and promotes good design practices such as loose coupling and separation of concerns.

Key features include:

* Inversion of Control (IoC)
* Dependency Injection (DI)
* Aspect-Oriented Programming (AOP)
* Data Access and Transaction Management
* Model-View-Controller (MVC) Web Applications
* Integration with ORM frameworks

**Reference:** [Spring Framework Overview](https://docs.spring.io/springframework/docs/3.2.x/spring-framework-reference/html/overview.html)

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

* **Inversion of Control (IoC):** IoC is a design principle where the control of object creation and lifecycle is transferred from the application code to the Spring container.
* **Dependency Injection (DI):** DI is a technique in which an object’s dependencies are provided by an external source (the IoC container) rather than the object itself creating them.

Spring supports DI through:

* XML configuration
* Annotation-based configuration
* Java-based configuration

**Reference:** [Baeldung: IoC and DI in Spring](https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring" \t "_blank)

**Spring Modules**

Spring is organized into several modules, including:

* **Core Container:** Provides the fundamental IoC and DI features.
* **AOP (Aspect-Oriented Programming):** Allows modularization of cross-cutting concerns.
* **Data Access/Integration:** JDBC, ORM (Hibernate, JPA), JMS, Transactions.
* **Web:** MVC, Web WebSocket, Web Portlet.
* **Test:** Support for testing Spring components.

**Benefits of Using Spring in Java Applications**

* **Loose Coupling:** Through DI, Spring promotes loosely coupled code.
* **Modularity:** Applications can use only the modules they need.
* **Testability:** Components are easier to test due to dependency injection.
* **Integration:** Seamless integration with various technologies (JPA, Hibernate, JMS, etc.).
* **Community and Documentation:** Well-documented and widely supported.

**Setting up a Spring Project with Maven**

* **Maven** is a popular build automation and dependency management tool for Java.
* To create a Spring project:
  1. Create a new Maven project.
  2. Add Spring dependencies in the pom.xml file.
  3. Use Maven to build and manage your project’s dependencies.

**Reference:** [StudyTonight: Spring Maven Project](https://www.studytonight.com/springframework/spring-maven-project" \t "_blank)

**Spring IoC Container**

* The **IoC Container** is the core of the Spring Framework, responsible for instantiating, configuring, and assembling beans.
* **BeanFactory** and **ApplicationContext** are two main types of IoC containers.
  + ApplicationContext is a superset of BeanFactory with more features.
* Beans and their dependencies can be defined using XML, annotations, or Java configuration.

**Reference:** [Baeldung: IoC and DI in Spring](https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring" \t "_blank)

**Spring Bean Configuration**

* **XML Configuration:** Define beans and dependencies in XML files.
* **Annotation-Based Configuration:** Use annotations like @Component, @Service, @Repository, and @Controller for component scanning.
* **Java-Based Configuration:** Use @Configuration and @Bean annotations to define beans in Java classes.
* **Mixing Configurations:** XML and Java-based configurations can be combined as needed.

**Reference:** [Baeldung: Spring Bean Annotations](https://www.baeldung.com/spring-bean-annotations" \t "_blank)

**Dependency Injection in Spring**

* **Constructor Injection:** Dependencies are provided through a class constructor.
* **Setter Injection:** Dependencies are set through JavaBean setter methods.
* **Autowiring:** Spring can automatically resolve and inject collaborating beans.
* **Qualifiers:** Use @Qualifier to resolve conflicts when multiple beans of the same type exist.
* **@Resource and @Inject:** Additional annotations for dependency injection.

**Reference:** [Spring Docs: Dependency Injection](https://docs.spring.io/springframework/reference/core/beans/dependencies/factory-collaborators.html)

**Spring AOP (Aspect-Oriented Programming)**

* **AOP** allows the separation of cross-cutting concerns (e.g., logging, security) from business logic.
* **Aspect:** A module that encapsulates a concern.
* **Advice:** Action taken at a join point (before, after, around).
* **Pointcut:** Specifies where advice should be applied.
* **Joinpoint:** Points in the program execution (e.g., method call).
* **Proxying:** Spring uses proxies to implement AOP features.

**Reference:** [GeeksforGeeks: AOP in Spring](https://www.geeksforgeeks.org/aspectoriented-programming-and-aop-in-springframework/" \t "_blank)

**Spring MVC and ORM**

* **Spring MVC:** A web framework based on the Model-View-Controller design pattern.
  + **Controller Layer:** Handles user requests.
  + **Model Layer:** Represents application data.
  + **View Layer:** Renders the output (JSP, Thymeleaf, etc.).
  + Features: Form handling, validation, exception handling.
* **ORM Integration:** Spring supports ORM frameworks like Hibernate and JPA for data persistence.

**Reference:** [GeeksforGeeks: Spring MVC](https://www.geeksforgeeks.org/springmvc-framework/" \t "_blank)

**Spring Boot (Introduction)**

* **Spring Boot** simplifies Spring application development by providing:
  + Auto-configuration
  + Embedded servers
  + Standalone applications
  + Convention over configuration
* **Creating a Spring Boot Application:**
  + Add spring-boot-starter dependencies.
  + Use @SpringBootApplication annotation.
  + Run the application with a simple main() method.

**Reference:** [GeeksforGeeks: Spring Boot](https://www.geeksforgeeks.org/springboot/" \t "_blank)