Spring Framework

https://spring.io/why-spring

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

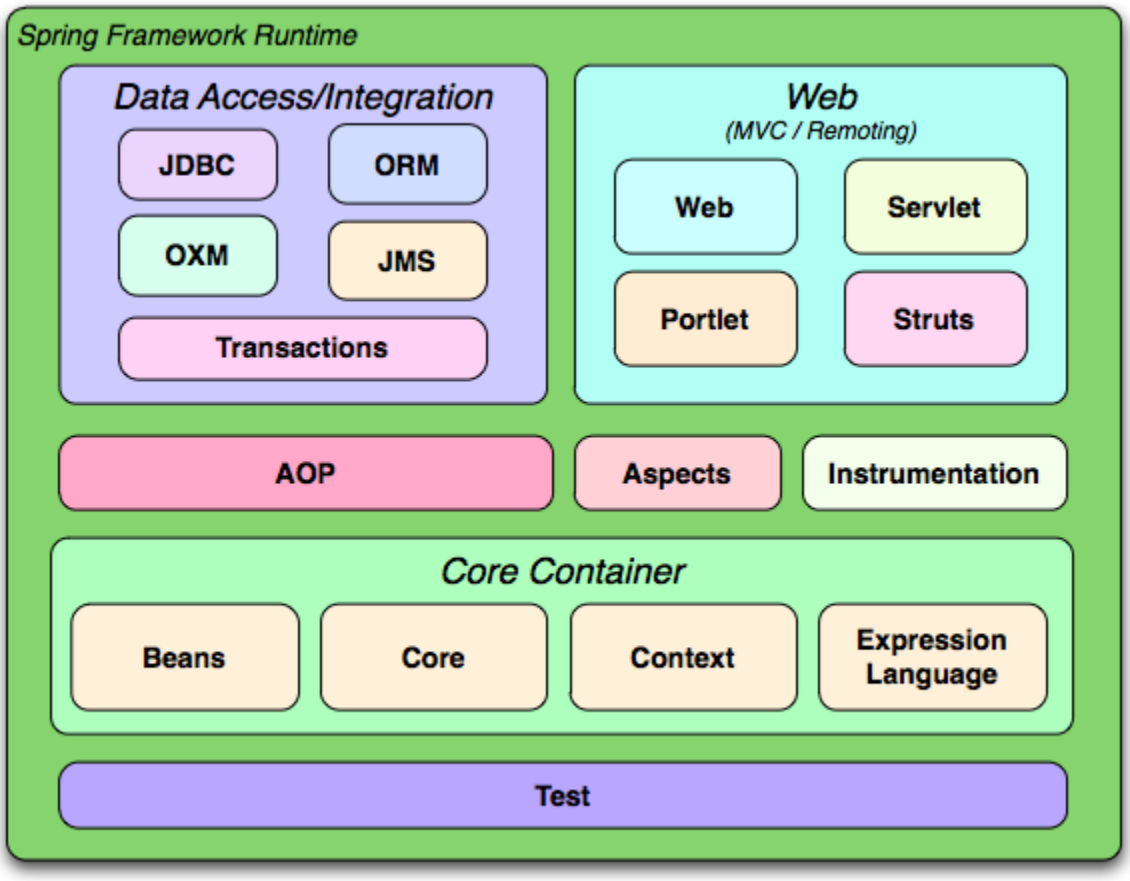
https://docs.spring.io/spring-framework/reference/index.html

https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/overview.html

1. Introduction to Spring Framework

- **What is Spring?** Spring is a **Java framework** that helps developers build applications more easily by providing tools for managing the complexity of large projects. Think of Spring as a helper that takes care of many routine tasks, allowing you to focus on the business logic.

The Spring Framework is widely used because it simplifies Java development, offering speed, flexibility, and productivity. It provides a comprehensive ecosystem for building applications, from microservices and web apps to cloud-native solutions. Spring's core concepts like Inversion of Control (IoC) and Dependency Injection (DI) streamline code management, while tools like Spring Boot make project setup fast. Its security features, vast third-party integrations, and strong community support make Spring a powerful framework for modern development.



2. Spring Core Concepts

Inversion of Control (IoC) and Dependency Injection (DI) are core concepts in the Spring Framework that make applications more flexible and easier to manage.

Inversion of Control (IoC):

- IoC is a design principle that means the control of creating and managing objects (like services, components, etc.) is shifted from the application to the Spring Framework.
- In simpler terms, instead of the application controlling how objects are created and used, the Spring Framework takes care of it. The framework "inverts" this control.

  Normally, you might manually create an object like this:

  Car myCar = new Car();

  With IoC, Spring creates the object for you:

  Car myCar = context.getBean(Car.class); // Spring handles object creation

Dependency Injection (DI):

- DI is a specific way to implement IoC, where the dependencies (like objects or services that a class needs) are "injected" into a class rather than the class creating them on its own.
- This makes the code more modular and easier to maintain. Instead of hardcoding dependencies inside a class, Spring injects them automatically when needed.
- Types of Dependency Injection:

1. Constructor Injection: Dependencies are passed through the constructor.

2. Setter Injection: Dependencies are injected through setter methods.

Without DI (the class creates its own dependencies):

```
class Car {
   private Engine engine = new Engine(); // Car creates the Engine
}
```

With DI (injects the Engine object):

```
class Car {
   private Engine engine;
   // Constructor Injection
   public Car(Engine engine) {
      this.engine = engine; // Engine is provided, not created inside Car
```

```
    }
}
```

How it works in Spring:

In Spring, you typically use annotations like `@Autowired` to tell the framework where to inject dependencies. Spring then handles the wiring of objects behind the scenes.

Benefits:

- Loose Coupling: Classes don't need to know about the specifics of how their dependencies are created. This makes them easier to change and maintain.
- Testability: Since dependencies are injected, you can easily swap them out with mock objects during testing.

In summary, IoC gives control of object creation to Spring, while DI injects the required dependencies into a class, making the application more flexible and easier to maintain.

What is a Bean in Spring?

A Bean is an object that is instantiated, assembled, and managed by the Spring IoC (Inversion of Control) container. Essentially, a bean is a simple Java object (POJO) that Spring creates and controls throughout its lifecycle.

Bean Scopes in Spring:

Bean scopes determine the lifecycle and visibility of beans within the Spring container. The common scopes are:

1. Singleton (default): Only one instance of the bean is created per Spring container.

2. Prototype: A new bean instance is created each time it is requested.

3. Request: One bean instance per HTTP request (for web applications).

4. Session: One bean instance per HTTP session (for web applications).

5. Application: One bean instance per ServletContext.

6. WebSocket: One bean instance per WebSocket connection.

These scopes allow developers to control how beans are created and used within an application.

3. Spring Configuration

Spring offers two main ways to configure your application: XML-based and Java-based. The modern and recommended way is Java-based configuration.

- Java Configuration: You use annotations and classes instead of XML files to define your beans.

```
@Configuration // Marks this class as a configuration class
public class AppConfig {
   @Bean // This method returns a bean managed by Spring
   public Car car() {
      return new Car(new Engine());
   }
}
```

- **Profiles**: Spring allows you to create different configurations for different environments (development, production, etc.) using **profiles**.

Example:

```
@Profile("dev") // This bean will only be active in the "dev" profile
@Bean
public Car devCar() {
   return new Car(new DevEngine());
}
```

---

4. Spring AOP (Aspect-Oriented Programming)

Spring AOP (Aspect-Oriented Programming) is a programming paradigm that helps modularize cross-cutting concerns (code that is scattered throughout various modules, like logging, security, or transaction management). Instead of putting this code in every class, you can define it as an aspect and apply it where needed, without modifying the core logic.

Key Concepts:

- Aspect: A module containing cross-cutting concerns (e.g., logging).

- Join Point: A point in the execution of a program (e.g., method execution).
- Advice: The action taken by an aspect at a join point (e.g., logging before a method runs).
- Pointcut: Defines when and where advice should apply (e.g., "apply advice to all methods in a class").
- Weaving: Linking aspects with target objects to create an advised object.

Why AOP?

AOP helps keep your core business logic clean and focused, while handling secondary concerns (like logging or security) separately. This results in better separation of concerns and more maintainable code.

Example:

Instead of adding logging to every method, you define a logging aspect and specify where it should be applied:

```
@Aspect
public class LoggingAspect {
   @Before("execution(* com.example..*(..))")
   public void logBeforeMethod() {
      System.out.println("Method called!");
   }
}
```

This logging aspect will run before the methods matching the specified pointcut (`execution(* com.example..*(..))`).

---

5. Spring Data Access

Spring Data Access simplifies database interactions, allowing you to work with various data sources like relational databases (e.g., MySQL, PostgreSQL) and NoSQL databases (e.g., MongoDB) easily.

Key components:

- JdbcTemplate: Simplifies working with JDBC by reducing boilerplate code for querying and managing database connections.
- Spring Data JPA: Provides an abstraction over JPA (Java Persistence API) and ORM tools like Hibernate, enabling easy CRUD operations with minimal configuration.

- Transaction Management: Allows declarative management of transactions with the `@Transactional` annotation for controlling commits and rollbacks.

---

6. ApplicationContext :

It is the central interface to provide configuration information to the Spring Framework. It manages the lifecycle of beans and enables features like dependency injection, event handling, and resource loading. It extends the BeanFactory interface (which only manages bean instantiation) by offering more enterprise-specific functionalities, such as handling message resources, application events, and AOP support.

Common implementations of `ApplicationContext` include:

- ClassPathXmlApplicationContext

- AnnotationConfigApplicationContext

It is essential for managing and configuring beans within a Spring-based application.

---

7. Spring Expression Language (SpEL):

It is a powerful expression language in the Spring Framework that allows you to dynamically access and manipulate object properties, arrays, lists, maps, methods, and more within Spring-managed beans.

Key Features:

- Expression evaluation: Evaluate expressions like arithmetic operations, method calls, etc.
- Property access: Access object properties (`person.name`).
- Method invocation: Call methods (`person.getName()`).
- Conditional operators: Use logical and comparison operators.
- Collections: Access and manipulate collections (`list[0]`, `map['key']`).

SpEL is commonly used in annotations like `@Value`, `@PreAuthorize`, and within XML or Java configurations for dynamic behavior.

---

8. Spring Security

Spring Security helps you add **authentication** (like login) and **authorization** (permissions) to your application.

Spring Security is a robust framework that provides authentication, authorization, and protection against common attacks for Spring applications. It integrates easily with Spring applications to secure web, API, and other services.

Key Features:

- Authentication and Authorization: Secures user access with roles and permissions.
- Integration with OAuth2/JWT: Supports modern security protocols.
- Method-level security: Secure individual methods using annotations like `@PreAuthorize`.
- Protection against attacks: Defends against CSRF, XSS, and other threats.
- It allows for customizable and flexible security configurations.
- Example of securing a web application:

---

9. Spring Testing

- **Unit Testing**: Use **JUnit** and **Mockito** to write unit tests for your Spring application.

Spring annotations:

1. @Component: Marks a class as a Spring-managed component. (part of DI).

2. @Autowired: Injects dependencies automatically (part of DI).

3. @Bean: Defines a Spring-managed bean method. (part of DI).

4. @SpringBootApplication: Marks the main class for Spring Boot applications.

5. @Controller: Designates a class as a web controller.

6. @RequestMapping: Maps web requests to specific handler methods.

7. @GetMapping: Maps HTTP GET requests.

8. @PostMapping: Maps HTTP POST requests.

9. @ModelAttribute: Binds method parameters to model attributes.

10. @PathVariable: Binds URL variables to method parameters.

11. @Repository: Marks a class as a DAO (Data Access Object).

12. @Service: Marks a service layer class. (part of DI).

13. @Configuration: Marks class as a configuration class