Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

Spring

Study

ARNIKA PATEL
CSE, PIT
Parul University

5.1 Introduction to Spring

- Spring Framework is a comprehensive and versatile platform for enterprise Java development. It is known for its **Inversion of Control (IoC)** and **Dependency Injection (DI)** capabilities that simplify creating modular and testable applications.
- Key features include **Spring MVC** for web development, **Spring Boot** for rapid application setup, and **Spring Security** for robust authentication and authorization.
- With a rich ecosystem covering **Spring Data** for database interactions and **Spring Cloud** for building microservices, **Spring** supports scalable and resilient enterprise solutions, making it an essential framework for developers of all experience levels.
- Spring framework is a Java platform that is open source. Rod Johnson created it, and it was first released under the Apache 2.0 license in June 2003. When it comes to size and transparency, Spring is a featherweight. Spring framework's basic version is about 2MB in size.
- **Applications of Spring**
    - **POJO Based:** Spring allows developers to use POJOs to create enterprise-class apps
    - **Modular** : Spring is set up in a modular approach.
    - **Integration with Existing Frameworks:** Spring does not reinvent the wheel; rather, it makes extensive use of existing technologies such as numerous ORM frameworks, logging frameworks, JEE, Quartz, and JDK timers, and other view technologies.
    - **Testability:** Because environment-dependent code is put into this framework, testing a Spring-written application is trivial.
    - **Web MVC:** Spring's web framework is a well-designed web MVC framework that is an excellent alternative to web frameworks like Struts and other over-engineered or less popular web frameworks.
    - **Central Exception Handling:** Spring provides a handy API for converting technology-specific exceptions (such as those raised by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.
    - **Lightweight:** IoC containers are typically lightweight, especially when compared to EJB containers, for example.

5.2 Spring Architecture

- The Spring framework is a widely used open-source Java framework that provides a comprehensive programming and configuration model for building enterprise applications.
- Its architecture is designed around two core principles:
    1. Dependency Injection (DI)
    2. Aspect-Oriented Programming (AOP)

Dependency Injection (DI)

- DI is a design pattern that promotes loose coupling between components by allowing objects to be injected into a class rather than the class creating them itself. The Spring framework provides an Inversion of Control (IoC) container which is responsible for managing these dependencies.
- The BeanFactory and ApplicationContext are the core interfaces of the IoC container.
    - BeanFactory provides basic dependency injection.
    - ApplicationContext adds advanced features like internationalization, event propagation, and resource loading.
- Spring manages the lifecycle of beans, including initialization, destruction, and scopes (e.g., singleton, prototype, request, session).
- **Need for Dependency Injection:**
    - Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be **dependent** on class Two. Now though it might appear okay to depend on a module on the other, in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided. Spring IOC resolves such dependencies with Dependency Injection, which makes the code **easier to test and reuse** .
    - Loose coupling between classes can be possible by defining interfaces for common functionality and the injector will instantiate the objects of required implementation. The task of instantiating objects is done by the container according to the configurations specified by the developer.
- **Types of Spring Dependency Injection:** There are two primary types of Spring Dependency Injection:

  1. Setter Dependency Injection (SDI): Setter DI involves injecting dependencies via setter methods. To configure SDI, the @Autowired annotation is used along with

setter methods, and the property is set through the <property> tag in the bean configuration file.

2. Constructor Dependency Injection (CDI): Constructor DI involves injecting dependencies through constructors. To configure CDI, the <constructor-arg> tag is used in the bean configuration file.

Aspect-Oriented Programming (AOP)

- AOP allows developers to modularize cross-cutting concerns such as logging, security, and transaction management.
- These concerns can be applied across multiple components without modifying the core business logic.
- Spring AOP uses advice, pointcuts, and aspects to implement cross-cutting concerns. For example, you can use AOP to log method execution times or enforce security checks

Overall, the Spring framework architecture is based on the principles of modularity, separation of concerns, and flexibility, providing developers with a powerful set of tools to build robust, scalable, and maintainable enterprise applications.

5.3 Spring Modules

The Spring framework consists of several modules, which can be categorized into four main areas:

1. **Core Container:** The Core Container provides the fundamental functionality of the Spring framework, including the IoC container and ApplicationContext.
    - **Spring Core:** This module provides the fundamental functionality of the Spring framework, including IoC and DI. The IoC container is the heart of the Spring Framework, responsible for creating and managing instances of JavaBeans. It uses dependency injection to wire the beans together.
    - **Spring Beans:** This module provides the BeanFactory, which is the basic building block of the IoC container, and the BeanWrapper, which is responsible

for managing the lifecycle of a bean. The Bean Factory is the core interface for accessing the IoC container. It provides methods for retrieving beans.

- **Spring Context:** This module provides the ApplicationContext, which is an advanced version of the BeanFactory and provides additional features, such as internationalization and resource loading, and the ability to publish and consume events.

- **Spring Expression Language (SpEL):** This module provides a powerful expression language for querying and manipulating objects during runtime. SpEL supports a wide range of features, including property access, method invocation, conditionals, loops, and type conversion.

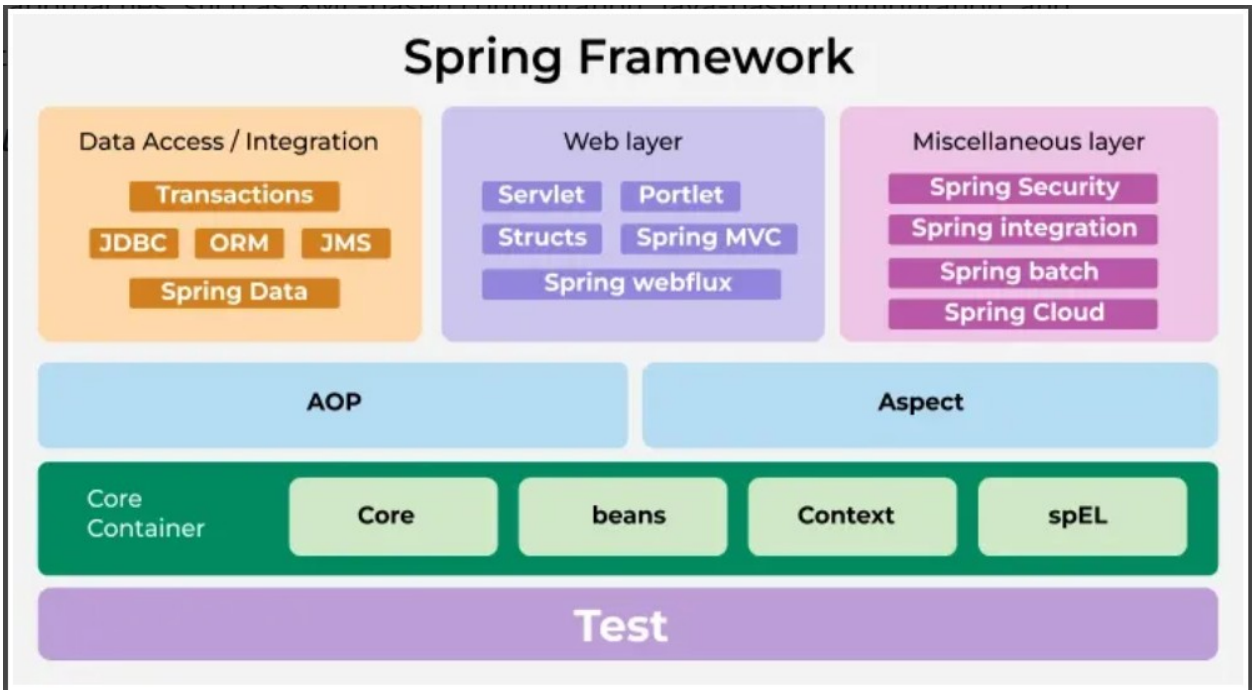

Fig. 5.1 Spring Modules

2. **Data Access/Integration:** The Data Access/Integration area provides support for integrating with databases and other data sources.

    - **Spring ORM:** Spring ORM provides a higher-level abstraction layer on top of ORM frameworks, allowing developers to write less boilerplate code and more easily integrate ORM technologies with other Spring features, such as transaction management and caching.

- **Spring Data:** This module provides a consistent and easy-to-use programming model for working with data access technologies, including databases, NoSQL, and cloud-based data services.

- **Spring Transaction:** This module provides support for declarative transaction management in Spring applications. Spring Transaction provides support for various transaction propagation and isolation levels, allowing developers to manage transactions at different levels of granularity.

3. **Web:** The Web area provides support for building web applications, including the Spring MVC and Spring WebFlux modules.

    - **Spring MVC:** This module provides a Model-View-Controller (MVC) framework for building web applications. Spring MVC provides a range of features, including support for handling HTTP requests and responses, form handling, data binding, validation, and more.

    - **Spring WebFlux:** This module provides a reactive programming model for building web applications that require high concurrency and scalability.

    - **Spring Web Services:** This module provides support for building SOAP-based and RESTful web services. Spring Web Services provides support for generating WSDL (Web Services Description Language) from Java classes, and for generating Java classes from WSDL.

4. **Miscellaneous:** The Miscellaneous area includes other modules that provide additional functionality, such as the Spring Security module for authentication and authorization features.

    - **Spring Security:** This module provides authentication and authorization features for Spring applications. Spring Security provides a range of authorization mechanisms, such as role-based access control and expression-based access control.

    - **Spring Integration:** This module provides support for building message-driven and event-driven architectures.

- **Spring Batch:** This module provides support for batch processing and integration with enterprise systems.

- **Spring Cloud:** This module provides support for building cloud-native applications using Spring technologies.

5.4 Autowiring

- Autowiring in the Spring framework can inject dependencies automatically. The Spring container detects those dependencies specified in the configuration file and the relationship between the beans. This is referred to as Autowiring in Spring.

- To enable Autowiring in the Spring application we should use @Autowired annotation. Autowiring in Spring internally uses constructor injection. An autowired application requires fewer lines of code comparatively but at the same time, it provides very little flexibility to the programmer.

Modes of Autowiring

1. No

This mode tells the framework that autowiring is not supposed to be done. It is the default mode used by Spring.

```
<bean id="state" class="sample.State">
        <property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City"></bean>
```

2. byName

It uses the name of the bean for injecting dependencies. However, it requires that the name of the property and bean must be the same. It invokes the setter method internally for autowiring.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City" autowire="byName"></bean>
```

3. byType

It injects the dependency according to the type of the bean. It looks up in the configuration file for the class type of the property. If it finds a bean that matches, it injects the property. If not,

the program throws an error. The names of the property and bean can be different in this case. It invokes the setter method internally for autowiring.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City" autowire="byType"></bean>
```

4. constructor

It injects the required dependencies by invoking the constructor. It works similar to the "byType" mode but it looks for the class type of the constructor arguments. If none or more than one bean are detected, then it throws an error, otherwise, it autowires the "byType" on all constructor arguments.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City" autowire="constructor"></bean>
```

5. autodetect

The autodetect mode uses two other modes for autowiring - constructor and byType. It first tries to autowire via the constructor mode and if it fails, it uses the byType mode for autowiring. It works in Spring 2.0 and 2.5 but is deprecated from Spring 3.0 onwards.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City" autowire="autodetect"></bean>
```

Advantage of Autowiring: It requires less code because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring: No control of the programmer and It can't be used for primitive and string values.

Table 5.1 modes of autowiring

Modes	Description
No	This mode tells the framework that autowiring is not supposed to be done. It is the default mode used by Spring.
byName	It uses the name of the bean for injecting dependencies.
byType	It injects the dependency according to the type of bean.

Constructor	It injects the required dependencies by invoking the constructor.
Autodetect	The autodetect mode uses two other modes for autowiring - constructor and byType.

5.5 Application Context

- ApplicationContext belongs to the Spring framework.
- Spring IoC container is responsible for instantiating, wiring, configuring, and managing the entire life cycle of beans or objects.
- BeanFactory and ApplicationContext represent the Spring IoC Containers.
- ApplicationContext is the sub-interface of BeanFactory. It is used when we are creating an enterprise-level application or web application. ApplicationContext is the superset of BeanFactory, whatever features provided by BeanFactory are also provided by ApplicationContext.
- **ApplicationContext Features**
    - ApplicationContext provides basic features in addition to enterprise-specific functionalities which are as follows:
    - Publishing events to registered listeners by resolving property files.
    - Methods for accessing application components.
    - Supports Internationalization.
    - Loading File resources in a generic fashion.
- **ApplicationContext Implementation Classes**
    1. **AnnotationConfigApplicationContext container**
        - AnnotationConfigApplicationContext class was introduced in Spring 3.0. It accepts classes annotated with @Configuration, @Component, and JSR-330 compliant classes. The constructor of AnnotationConfigApplicationContext accepts one or more classes.

ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class, AppConfig1.class);

2. **AnnotationConfigWebApplicationContext**

- AnnotationConfigWebApplicationContext class was introduced in Spring 3.0. It is similar to AnnotationConfigApplicationContext for a web environment. It accepts classes annotated with @Configuration, @Component, and JSR-330 compliant classes. These classes can be registered via ***register() method*** or passing base packages to ***scan() method***.

// Implementing WebApplicationInitializer

public class MyWebApplicationInitializer implements WebApplicationInitializer {

// Servlet container

public void onStartup(ServletContext container) throws ServletException {

AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext(); context.register(AppConfig.class); context.setServletContext(container);

// Servlet configuration

} }

3. **XmlWebApplicationContext**

- Spring MVC Web-based application can be configured completely using XML or Java code. Configuring this container is similar to the AnnotationConfigWebApplicationContext container, which implies we can configure it in web.xml or using java code**.**

XmlWebApplicationContext context = new XmlWebApplicationContext(); context.setConfigLocation("/WEB-INF/spring/applicationContext.xml"); context.setServletContext(container);

4. **FileSystemXmlApplicationContext**

   - FileSystemXmlApplicationContext is used to load XML-based Spring Configuration files from the file system or from URL. We can get the application context using Java code. It is useful for standalone environments and test harnesses. The following code shows how to create a container and use the XML as metadata information to load the beans.

     ApplicationContext context = new FileSystemXmlApplicationContext(path);

5. **ClassPathXmlApplicationContext**

   - FileSystemXmlApplicationContext is used to load XML-based Spring Configuration files from the classpath. We can get the application context using Java code. It is useful for standalone environments and test harnesses. The following code shows how to create a container and use the XML as metadata information to load the beans.

     ApplicationContext context = new ClassPathXmlApplicationContext("applicationcontext/student-bean-config.xml");

5.6 Spring- Annotation based Configuration

- Starting from Spring 2.5 it became possible to configure the dependency injection using **annotations**. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

- Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.

- Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider the following configuration file in case you want to use any annotation in your Spring application.
  <beans…>

```
    <context:annotation-config/>
    <!-- bean definitions go here -->
</beans>
```

- Once <context:annotation-config/> is configured, you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors.

Table 5.2 Annotation

Sr.No	Annotation & Description
1	**@Required** The @Required annotation applies to bean property setter methods.
2	**@Autowired** The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.
3	**@Qualifier** The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
4	**JSR-250 Annotations** Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

References:

1. **Book Reference**

   Jim Farley, William Crawford, David Flanagan. Java Enterprise in a Nutshell, O'Reilly

2. **Book Reference**

   Rocha, R., Purificação, J. (2018). Java EE 8 Design Patterns and Best Practices: Build Enterprise-ready Scalable Applications with Architectural Design Patterns. Germany: Packt Publishing..

3. **Website Reference**

   https://www.scribd.com/document/268349254/Java-8-Programming-Black-Book .

4. **Sources**

   https://developers.redhat.com/topics/enterprise-java

5. **Article**

   https://www.researchgate.net/publication/276412369_Advanced_Java_Programming

NAAC GRADE A++