**Spring Framework**

**->** Spring is a dependency injection framework is to make loosely coupled application.

-> We can develop standalone & enterprise application.

-> Spring framework makes the easy development of Java EE application.

-> It was developed by Rod Johnson in 2003.

**Spring Advantages**

**Modular and lightweight:**

Allowing developers to use only the components they need, making applications more lightweight and easier to maintain. This modularity also promotes better code organization

**Flexible Configuration:**

Including xml, java-based config., and annotation-based config. This flexibility allows developers to choose the suitable approach for their projects.

**DI:**

It simplifies the management of component dependencies, making code more testable and adaptable to changes.

**AOP:**

Allowing developers to separate cross cutting concerns like logging, security and transactions from the core application logic. This improves code modularity and maintainability.

**Simplified DB access:**

Reduce boilerplate code, improves data efficiency

**Security:**

Provides a robust framework for implementing authentication & authorization, making it easier to secure web applications and services.

**Spring Container:**

Heart of spring i.e. core (Jase JVM java application ko handle karta, usi tarah se spring spring applications ko handle karta)

**Responsibility**:

1. **Manage bean objects**

* A Java bean class is just a standard. Often used to encapsulate data and adhere to java beans conventions for easy integration with frameworks. It’s a regular java class, except it follows certain conventions:

1. All properties are private (use getters & setters)
2. A public no argument constructor
3. Implements serializable

The object created of this java bean class is known as Bean Object.

* **POJO Class:**

1. A simple java class with fields and getters/setters, used for data representation without framework dependencies
2. **Manage bean life-cycle**
3. **Dependency Injection**
4. **AOP**
5. **Transaction Management**
6. **I18N (Internalization) etc.**

**IOC Container**

* The IoC container is responsible to instantiate, configure, and assemble the objects.
* The IoC container gets information from the XML file and works accordingly. The main tasks performed by IoC containers are:

1. To instantiate the application class
2. To configure the object
3. To assemble the dependencies between the objects.

* There are two types of IoC containers. They are:

1. BeanFactory (It became old)
2. ApplicationContext (we use this now): - It’s an interface
3. Application context (Spring container) is an interface in spring which is used to manage beans, handle application events, and access resources
4. Some implemented classes of Application context are:
5. ClassPathApplicationContext (Used for XML config.)
6. AnnotationConfigApplicationContext (Used for Java config)

Spring Container

 

 



Different bean Objects dklkslfksklfkfbeanObjectsbbeecs

**Configuration Metadata**

1. XML- file config
2. Java file-config
3. Annotation-based config

* **Java File-Config:**

1. @Configuration: indicates that the class is spring configuration class. It means that annotated java class contains beans definitions and configuration setting for the spring application context.
2. @Configuration allows us to define beans and their dependencies in a Java-based way instead of using XML config. Files.
3. In xml-based spring config, we use <bean> tag to define beans whereas in java-based config, we use methods annotated with @Bean to define beans
4. In xml config., we provide an “id” attribute to specify the bean name and specify the class name using “class” attribute whereas in java config., the “method name” becomes the default bean name, and the “return type” of the method determines the class of the object that will be created as the bean.

* **Annotation based config:**

1. If we are doing this config then still, we need to create either xml or java-based file.
2. @Component annotation (is also known as stereotype annotation) is used to declare a class as a Spring bean, which is a managed component in the Spring application context
3. It helps spring automatically detect and manage these beans during application startup, making them available for dependency injection and other spring features.
4. In a spring xml config file, <context: component-scan> is used to instruct the spring framework to perform component scanning.
5. The @Value annotation in spring is used to inject values into Spring bean fields or methods.
6. It scans specified packages in the application’s classpath for classes annotated with annotations like @Component, @Service, @Repository & @Controller. These discovered classes are then registered as Spring beans, enabling automatic dependency injection and other spring features.
7. With java file we need to use @Configuration @ComponentScan

**Lifecycle of Bean Object:**

1. **Loading bean definitions:**
2. Spring loads bean definitions from various sources such as XML configuration files, java-based config classes or component scanning
3. These bean definitions contain information about the bean class, dependencies and other configuration details.
4. **Bean Instantiation:**
5. After bean definitions are loaded, the spring container creates instances of beans based on these definitions.
6. This involves invoking the bean class constructor, either the default or a specified constructor, to create an actual instance of the bean.
7. The newly created bean instance is now ready for configuration and initialization.
8. **Bean initialization:**
9. Once the bean is instantiated, spring proceeds to configure and initialize it.
10. Property values are set using setters, constructor arguments, or field injection, populating the bean’s state.
11. **Use bean object:**
12. With the help of context (object of interface), we can access the object.
13. **Bean destruction:**
14. Bean object is destroyed or deleted.

**Dependency Injection:**

1. It is a design Pattern used in spring framework to achieve Inversion of Control (IoC).
2. Its main task is to “inject” the “dependency” means inject one object into another object.
3. It’s used to achieve loose coupling.

We can achieve DI in two ways:

1. Setter method DI b.) Constructor DI

We can achieve these DI Patterns with two ways:

1. XML File
2. Java File
3. Setter Method DI:

In xml file when we inject object into another, then we use “ref” attribute to reference that bean “id”

In Java Bean/POJO we don’t create constructor, only create setter methods.

<bean class=” FULL\_PATH” id=” #“>

<property name = “ “ value = “ “ />

<property name = “ “ value = “ “ />

</bean>

<bean class = “FULL\_PATH “ id = “ “ >

<property name = “ “ ref = “ID\_OF\_INJECTED\_OBJECT“ />

</bean>

1. Constructor DI:

In xml file we will not use property tags we will use <constructor-arg value =” #”/>

Here also we will use “ref” attribute if we need to inject the object of other class.

In CDI Java bean/POJO will not have setter methods , only have constructors.

<bean class=” FULL\_PATH” id=” #“>

< constructor-arg value = “ “ />

<constructor-arg value = “ “ />

</bean>

<bean class = “FULL\_PATH “id = “ “ >

<constructor-arg ref = “ID\_OF\_INJECTED\_OBJECT“ />

</bean>

|  |  |
| --- | --- |
| Setter Method DI: - | Constructor DI: - |
| 1. Dependencies are injected into a class through setter methods. | **Dependencies are injected into a class through Constructor.** |
| 1. Setter method DI is more readable | **Constructor DI is less readable** |
| 1. Setter method DI is more flexible | **Constructor DI is less flexible** |

**AUTOWIRING:**

1. Feature of Spring Framework by which we can achieve DI automatically.
2. The @Autowired annotation in Spring is used for automatic dependency injection.
3. It tells spring to take the bean and plug it where we need it
4. It lets spring to handle object connections for us.
5. Autowiring can be achieved by: -
6. Annotation-based Autowiring – Used @Autowired & @Qualifier
7. XML-based Autowiring: - Used “autowire” attribute
8. @Qualifier annotation in spring helps pick the right bean among multiple beans of the same type. It helps spring to know which bean you want to inject and resolves ambiguity.

**Disadvantage of Autowiring:**1. It can be achieved only on non-primitive or user-defined datatypes, not on primitive data types.