**Spring Core**

Topics to be covered

1. IoC container

2. Resources

3. Validation, Data binding & Type conversion

4. Spring Expression Language

5. Aspect Oriented Programming with Spring

6. Spring AOP APIs

7. Null-safety

8. Data buffers & Codecs

9. Logging

**1. IoC Container**

**1.1 Introduction to the Spring IoC Container & Beans**

> IoC: Inversion of Control

> IoC is also known as Dependency Injection (DI).

> It is a process whereby objects define their dependencies (i.e., the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.

> The container then injects those dependencies when it creates the bean.

> The process is fundamentally the inverse of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes or a mechanism such as the Service Locator pattern.

> The **org.springframework.beans** & **org.springframework.context** packages are the basis for Spring Framework’s IoC container.

> The **BeanFactory** interface provides an advanced configuration mechanism capable of managing any type of object. **ApplicationContext** is a sub-interface of **BeanFactory**. It adds:

* Easier integration with Spring’s AOP features
* Message resource handling (for use in internationalization)
* Event publication
* Application-layer specific contexts such as the **WebApplicationContext** for use in web applications.

> In Spring, the objects that form the backbone of your application & that are managed by the **Spring IoC container** are called **Spring Beans**.

> A **Spring bean** is an object that is instantiated, assembled, & managed by a **Spring IoC container**. Otherwise, a bean is simply one of many objects in your application.

> Beans, & the dependencies among them, are reflected in the configuration metadata used by a container.

**1.2 Container Overview**

> The **org.springframework.context.ApplicationContext** interface represents the Spring IoC container & is responsible for instantiating, configuring, & assembling the beans.

> The container gets its instructions on what objects to instantiate, configure, & assemble by reading configuration metadata.

> The configuration metadata is represented in XML, Java annotations, or Java code. It lets you express the objects that compose your application & the rich interdependencies between those objects.

> Some implementing classes of **ApplicationContext**: **ClassPathXmlApplicationContext, FileSystemXmlApplicationContext, XmlWebApplicationContext** etc.

> The following diagram shows a high-level view of how Spring works.



**Fig. The Spring IoC container**

> This configuration metadata represents how you, as an application developer, tell the Spring container to instantiate, configure, & assemble the objects in your application.

> 3 important methods to provide configuration metadata to the Spring container:

1. **XML based configuration**: We can specify configuration data in an XML file.
2. **Annotation-based configuration**: We can use Annotations to specify configuration. This was introduced in Spring 2.5
3. **Java-based configuration**: This is introduced from Spring 3.0. We can embed annotations like @Bean, @Import, @Configuration in java code to specify configuration metadata.

> Spring configuration consists of at least one & typically more than one bean definition that the container must manage.

> XML-based configuration metadata configures these beans as **<bean/> elements** inside a top-level **<beans/> element**.

> Java configuration typically uses **@Bean** annotated methods within a **@Configuration** class.

> Typically, you define service layer objects, data access objects (DAOs), presentation objects, infrastructure objects such as Hibernate, SessionFactories etc.

> Typically, one doesn’t configure fine-grained domain objects in the container, because it is usually the responsibility of DAOs & business logic to create and load domain objects.

> However, you can use Spring’s integration with AspectJ to configure objects that have been created outside the control of an IoC container.