Spring- Spring framework is an open-source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly. Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high-performing, easily testable, and reusable code.

Applications of Spring:

* POJO Based - Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
* Modular - Spring is organized in a modular fashion. Even though the number of packages and classes is substantial, you have to worry only about the ones you need and ignore the rest.
* Integration with existing frameworks - Spring does not reinvent the wheel, instead, it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz, and JDK timers, and other view technologies.
* Testability - Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.
* Web MVC - Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
* Central Exception Handling - Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
* Lightweight - Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
* Transaction management - Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework.

Architecture:



The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application.



The ApplicationContext container includes all functionality of the BeanFactorycontainer, so it is generally recommended over BeanFactory. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications where data volume and speed is significant.

Spring Bean Factory Container:

This is the simplest container providing the basic support for DI and defined by the org.springframework.beans.factory.BeanFactory interface. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for backward compatibility with a large number of third-party frameworks that integrate with Spring.

Spring ApplicationContext Container:

The Application Context is Spring's advanced container. Similar to BeanFactory, it can load bean definitions, wire beans together, and dispense beans upon request. Additionally, it adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by org.springframework.context.ApplicationContext interface.

The ApplicationContext includes all functionality of the BeanFactory, It is generally recommended over BeanFactory. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications.

Beans:

The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container.

The three important methods to provide configuration metadata to the Spring Container

* XML-based configuration file.
* Annotation-based configuration
* Java-based configuration

Singleton scope

If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object. The default scope is always singleton.

A bean definition can contain a lot of configuration information, including constructor arguments, property values, and container-specific information such as initialization method, static factory method name, and so on.

A child bean definition inherits configuration data from a parent definition. The child definition can override some values, or add others, as needed. Spring Bean definition inheritance can be defined as a parent bean definition as a template and other child beans can inherit the required configuration from the parent bean. When you use XML-based configuration metadata, you indicate a child bean definition by using the parent attribute, specifying the parent bean as the value of this attribute.

Dependency Injection:

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container "injects" objects into other objects or "dependencies". Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container. **Dependency Injection in Spring can be done through constructors, setters, or fields.**

The @Configuration annotation indicates that the class is a source of bean definitions. We can also add it to multiple configuration classes.

We use the @Bean annotation on a method to define a bean. If we don't specify a custom name, then the bean name will default to the method name.

The autowiring feature of the spring framework enables you to inject the object dependency implicitly. It internally uses a setter or constructor injection. Autowiring can't be used to inject primitive and string values. It works with reference only. The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property, or methods with arbitrary names and/or multiple arguments.

The @Resource annotation in spring performs the auto wiring functionality. This annotation follows the autowire=byName semantics in the XML-based configuration i.e. it takes the name attribute for the injection.

Aspect-Oriented Programming:

Aspect-Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing, declarative transactions, security, caching, etc.

**A**joinpoint**is a point during the execution of a program, such as the execution of a method or the handling of an exception.**

A Pointcut is a predicate that helps match Advice to be applied by an Aspect at a particular JoinPoint.

The Advice is often associated with a Pointcut expression and runs at any Joinpoint matched by the Pointcut.

Advice is an action taken by an aspect at a particular Joinpoint. Different types of advice include “around,” “before” and “after” advice.

SpringBoot:

Spring Boot is an open-source Java-based framework used to create a micro Service. It is developed by the Pivotal Team and is used to build stand-alone and production-ready spring applications.

Micro Service is an architecture that allows developers to develop and deploy services independently. Each service running has its process and this achieves the lightweight model to support business applications.

Spring Boot is designed with the following goals −

* To avoid complex XML configuration in Spring
* To develop production-ready Spring applications in an easier way
* To reduce the development time and run the application independently
* Offer an easier way of getting started with the application

The entry point of the spring boot application is the class that contains @SpringBootApplication annotation and the main method.