Spring Core Framework

1. How does Spring relates or differs from Java EE?

2. Why do we go for Spring Framework?

* Spring is a powerful open source, application framework created to reduce the complexity of enterprise application development. It is light-weighted and loosely coupled. It has layered architecture, which allows you to select the components to use, while also providing a cohesive framework for J2EE application development. Spring framework is also called framework of frameworks as it provides support to various other frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.
* Following are some of the major features of Spring Framework :
* **Lightweight:** Spring is lightweight when it comes to size and transparency.
* **Inversion of control (IOC):** The objects give their dependencies instead of creating or looking for dependent objects. This is called Inversion Of Control.
* **Aspect oriented Programming (AOP):** Aspect oriented programming in Spring supports cohesive development by separating application business logic from system services.
* **Container:**Spring Framework creates and manages the life cycle and configuration of the application objects.
* **MVC Framework:** Spring Framework’s MVC web application framework is highly configurable. Other frameworks can also be used easily instead of Spring MVC Framework.
* **Transaction Management:** Generic abstraction layer for transaction management is provided by the Spring Framework. Spring’s transaction support can be also used in container less environments.
* **JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers an exception hierarchy, which simplifies the error handling strategy.
* Advantages:
* With the [**Dependency Injection(DI)**](https://howtodoinjava.com/spring/spring-core/inversion-of-control-ioc-and-dependency-injection-di-patterns-in-spring-framework-and-related-interview-questions/) approach, dependencies are explicit and evident in constructor or JavaBean properties.
* 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.
* 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, other view technologies.
* Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.
* [**Testing an application**](https://howtodoinjava.com/junit/how-to-unit-test-spring-security-authentication-with-junit/) written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.
* 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.
* 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).

3. What is Spring configuration file?

There are three types of configuration.

A)XML based

A Spring configuration file is an XML file. This file mainly contains the classes information. It describes how those classes are configured as well as introduced to each other. The XML configuration files, however, are verbose and more clean. If it’s not planned and written correctly, it becomes very difficult to manage in big projects.

In Spring framework, dependencies and the services needed by beans are specified in configuration files, which are typically in an XML format. These configuration files usually start with <beans> tag and contain a lot of bean definitions AND application specific configuration options.

The main goal of Spring XML Configuration is to have all the Spring components configured by using xml files.  
This means that there will not be present any other type of Spring Configuration (like annotations or configuration via Java classes).

A Spring XML Configuration uses Spring namespaces to make available the sets of XML tags used in the configuration; the main Spring namespaces are: context, beans, jdbc, tx, aop, mvc, aso.

B)Java based

The central artifacts in Spring’s new Java-configuration support are @Configuration annotated classes and @Beanannotated methods.

The @Bean annotation is used to indicate that a method instantiates, configures and initializes a new object to be managed by the Spring IoC container. @Bean annotation plays the same role as the <bean/> element.

Annotating a class with @Configuration indicates that its primary purpose is as a source of bean definitions. Furthermore, @Configuration classes allow inter-bean dependencies to be defined by simply calling other @Beanmethods in the same class. The simplest possible @Configuration class would read as follows:

C)Annotation based

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 to have following configuration file in case you want to use any annotation in your Spring application.

4. What is BeanFactory?

5. What is Spring IoC container and what are its types?

At the core of the Spring Framework, lies the Spring container. The container creates the object, wires them together, configures them and manages their complete life cycle. The Spring container makes use of Dependency Injection to manage the components that make up an application. The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations or Java code.

The org.springframework.beans and org.springframework.context packages provide the basis for the Spring Framework’s IoC container. The BeanFactory interface provides an advanced configuration mechanism capable of managing objects of any nature. The ApplicationContext interface builds on top of the BeanFactory (it is a sub-interface) and adds other functionality such as easier integration with [**Spring’s AOP features**](https://howtodoinjava.com/category/frameworks/java-spring-tutorials/spring-aop/), [**message resource handling**](https://howtodoinjava.com/spring/spring-mvc/spring-mvc-internationalization-i18n-and-localization-i10n-example/) (for use in internationalization), event propagation, and application-layer specific contexts such as the WebApplicationContext for use in web applications.

The org.springframework.beans.factory.BeanFactory is the actual representation of the Spring IoC container that is responsible for containing and otherwise managing the aforementioned beans. The BeanFactory interface is the central IoC container interface in Spring.

6. What are the benefits of Inversion Of Control (IoC)?

Some of the benefits of IoC are:

* It will minimize the amount of code in your application.
* It will make your application easy to test because it doesn’t require any singletons or JNDI lookup mechanisms in your unit test cases.
* It promotes loose coupling with minimal effort and least intrusive mechanism.
* It supports eager instantiation and lazy loading of the services.

7. What are the common implementations of ApplicationContext?

The three commonly used implementation of 'Application Context' are

·**ClassPathXmlApplicationContext** It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code.  
ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");

·**FileSystemXmlApplicationContext** It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code.  
ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");

·**XmlWebApplicationContext** it loads context definition from an XML file contained within a web application.

8. What is the difference between BeanFactory and ApplicationContext?

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients. BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client. BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

1. A means for resolving text messages, including support for internationalization.
2. A generic way to load file resources.
3. Events to beans that are registered as listeners.

The three commonly used implementations of ApplicationContext are:

**ClassPathXmlApplicationContext** : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application’s classpath by using the code.

FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code.

XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

9. What is Dependency Injection (DI) in Spring?

In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don’t connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

Dependency injection is a pattern used to create instances of objects that other objects rely on without knowing at compile time which class will be used to provide that functionality. Inversion of control relies on dependency injection because a mechanism is needed in order to activate the components providing the specific functionality. Otherwise how will the framework know which components to create if it is no longer in control?

In Java, dependency injection may happen through 3 ways:

1. A constructor injection
2. A setter injection
3. An interface injection

10. What are the different types of dependency injection?

Spring supports 2 types of dependency injection, they are:

1) Constructor-based dependency injection: It is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

2) Setter-based dependency injection: It is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

11. Which type of DI would you suggest – Constructor-based or Setter-based?

Since you can mix both, Constructor- and Setter-based DI, it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies. Note that the use of a @Required annotation on a setter can be used to make setters required dependencies.

12. Explain how Spring resolves dependencies.

Spring sets properties and resolves dependencies as late as possible, when the bean is actually created. This means that a Spring container which has loaded correctly can later generate an exception when you request an object if there is a problem creating that object or one of its dependencies. For example, the bean throws an exception as a result of a missing or invalid property. This potentially delayed visibility of some configuration issues is why ApplicationContext implementations by default pre-instantiate singleton beans.

13. How do you declare inter bean dependencies?

One of the ways Spring recommends injecting inter-dependencies between beans is shown in the following sample copied from the Spring's reference guide :

@Configuration

public class AppConfig {

@Bean

public Foo foo() {

return new Foo(bar());

}

@Bean

public Bar bar() {

return new Bar("bar1");

So here, bean `foo` is being injected with a `bar` dependency.

However, there is one alternate way to inject dependency that is not documented well, it is to just take the dependency as a `@Bean` method parameter this way:

@Configuration

public class AppConfig {

@Bean

public Foo foo(Bar bar) {

return new Foo(bar);

}

@Bean

public Bar bar() {

return new Bar("bar1");

}

}

14. What are Spring Beans?

They are the objects that form the backbone of the user’s application.

* Beans are managed by the Spring IoC container.
* They are instantiated, configured, wired and managed by a Spring IoC container
* Beans are created with the configuration metadata that the users supply to the container
* .

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.

15. What are the different Bean scopes?

Singleton :-

This scopes the bean definition to a single instance per Spring IoC container (default).

prototype :-

This scopes a single bean definition to have any number of object instances.

request :-

This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

session :-

This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

global-session :-

This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.