What is Spring?

Spring is an open source development framework for Enterprise Java. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make Java EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

What are benefits of Spring Framework?

- **Lightweight:** Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.
- **Inversion of control (IOC):** Loose coupling is achieved in Spring, with the Inversion of Control technique. The objects give their dependencies instead of creating or looking for dependent objects.
- **Aspect oriented (AOP):** Spring supports Aspect oriented programming and separates application business logic from system services.
- **Container:** Spring contains and manages the life cycle and configuration of application objects.
- **MVC Framework:** Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks.
- **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (JTA).
- **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.

Explain the Core Container (Application context) module

This is the basic Spring module, which provides the fundamental functionality of the Spring framework. `BeanFactory` is the heart of any spring-based application. Spring framework was built on the top of this module, which makes the Spring container.

BeanFactory – BeanFactory implementation example

A `BeanFactory` is an implementation of the factory pattern that applies Inversion of Control to separate the application's configuration and dependencies from the actual application code.
The most commonly used `BeanFactory` implementation is the `XmlBeanFactory` class.

XMLBeanFactory

The most useful one is `org.springframework.beans.factory.xml.XmlBeanFactory`, which loads its beans based on the definitions contained in an XML file. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

Explain the JDBC abstraction and DAO module

With the JDBC abstraction and DAO module we can be sure that we keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. It provides a layer of meaningful exceptions on top of the error messages given by several database servers. It also makes use of Spring's AOP module to provide transaction management services for objects in a Spring application.

Explain the object/relational mapping integration module

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provides support to tie into several popular ORM frameworks, including Hibernate, JDO, and iBATIS SQL Maps. Spring's transaction management supports each of these ORM frameworks as well as JDBC.

Explain the web module

The Spring web module is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with Jakarta Struts.

Explain the Spring MVC module

MVC framework is provided by Spring for building web applications. Spring can easily be integrated with other MVC frameworks, but Spring's MVC framework is a better choice, since it uses IoC to provide for a clean separation of controller logic from business objects. With Spring MVC you can declaratively bind request parameters to your business objects.

What is Spring IoC container?

The Spring IoC is responsible for creating the objects,managing them (with dependency injection (DI)), wiring them together, configuring them, as also managing their complete lifecycle.

What are the benefits of IOC?

IOC or dependency injection minimizes the amount of code in an application. It makes easy to test applications, since no singletons or JNDI lookup mechanisms are required in unit tests. Loose coupling is promoted with minimal effort and least intrusive mechanism. IOC containers support eager instantiation and lazy loading of services.

What are the common implementations of the ApplicationContext?

The **FileSystemXmlApplicationContext** container loads the definitions of the beans from an XML file. The full path of the XML bean configuration file must be provided to the constructor.
The **ClassPathXmlApplicationContext** container also loads the definitions of the beans from an XML file. Here, you need to set`CLASSPATH` properly because this container will look bean configuration XML file in `CLASSPATH`.
The **WebXmlApplicationContext:** container loads the XML file with definitions of all beans from within a web application.

What is the difference between Bean Factory and ApplicationContext?

Application contexts provide a means for resolving text messages, a generic way to load file resources (such as images), they can publish events to beans that are registered as listeners. In addition, operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context. The application context implements `MessageSource`, an interface used to obtain localized messages, with the actual implementation being pluggable.

DEPENDENCY INJECTION

What is Dependency Injection in Spring?

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept, and it can be expressed in many different ways.This concept says that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

What are the different types of IoC (dependency injection)?

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

SPRING BEANS

What are Spring beans?

The Spring Beans are Java Objects that form the backbone of a Spring application. They are instantiated, assembled, and managed by the Spring IoC container. These beans are created with the configuration metadata that is supplied to the container, for example, in the form of XML definitions.

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named `"singleton"` if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

What does a Spring Bean definition contain?

A Spring Bean definition contains all configuration metadata which is needed for the container to know how to create a bean, its lifecycle details and its dependencies.

How do you provide configuration metadata to the Spring Container?

There are three important methods to provide configuration metadata to the Spring Container:

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

How do you define the scope of a bean?

When defining a in Spring, we can also declare a scope for the bean. It can be defined through the `scope` attribute in the bean definition. For example, when Spring has to produce a new bean instance each time one is needed, the bean's `scope` attribute to be `prototype`. On the other hand, when the same instance of a bean must be returned by Spring every time it is needed, the the bean `scope` attribute must be set to `singleton`.

Explain the bean scopes supported by Spring

There are five scoped provided by the Spring Framework supports following five scopes:

- In **singleton** scope, Spring scopes the bean definition to a single instance per Spring IoC container.
- In **prototype** scope, a single bean definition has any number of object instances.
- In **request** scope, a bean is defined to an HTTP request. This scope is valid only in a web-aware Spring ApplicationContext.
- In **session** scope, a bean definition is scoped to an HTTP session. This scope is also valid only in a web-aware Spring ApplicationContext.

- In **global-session** scope, a bean definition is scoped to a global HTTP session. This is also a case used in a web-aware Spring ApplicationContext.

The default scope of a Spring Bean is `Singleton`.

bean wiring

Wiring, or else bean wiring is the case when beans are combined together within the Spring container. When wiring beans, the Spring container needs to know what beans are needed and how the container should use dependency injection to tie them together.

bean auto wiring

The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans

Explain different modes of auto wiring?

The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

- **no:** This is default setting. Explicit bean reference should be used for wiring.
- **byName:** When autowiring `byName`, the Spring container looks at the properties of the beans on which `autowire` attribute is set to `byName` in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
- **byType:** When autowiring by `datatype`, the Spring container looks at the properties of the beans on which `autowire` attribute is set to `byType` in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.
- **constructor:** This mode is similar to `byType`, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
- **autodetect:** Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by `byType`.

Are there limitations with autowiring?

Limitations of autowiring are:

- **Overriding:** You can still specify dependencies using and settings which will always override autowiring.
- **Primitive data types:** You cannot autowire simple properties such as primitives, Strings, and Classes.
- **Confusing nature:** Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.
-
-

What is Spring Java-Based Configuration? Give some annotation example.

- Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.
An example is the `@Configuration` annotation, that indicates that the class can be used by the Spring IoC container as a source of bean definitions. Another example is the`@Bean` annotated method that will return an object that should be registered as a bean in the Spring application context.

What is Annotation-based container configuration?

An alternative to XML setups is provided by annotation-based configuration which relies on the bytecode metadata for wiring up components instead of angle-bracket declarations. Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class

@Required annotation

This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring. The container throws `BeanInitializationException` if the affected bean property has not been populated.

@Autowired annotation

The `@Autowired` annotation provides more fine-grained control over where and how autowiring should be accomplished. It can be used to autowire bean on the setter method just like `@Required` annotation, on the constructor, on a property or pn methods with arbitrary names and/or multiple arguments.

@Qualifier annotation

When there are more than one beans of the same type and only one is needed to be wired with a property, the `@Qualifier` annotation is used along with `@Autowired` annotation to remove the confusion by specifying which exact bean will be wired.

Spring DAO support

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows us to switch between the persistence technologies fairly easily and to code without worrying about catching exceptions that are specific to each technology.

ORM's Spring support

Spring supports the following ORM's:

- Hibernate
- iBatis
- JPA (Java Persistence API)
- TopLink
- JDO (Java Data Objects)
- OJB

SPRING MODEL VIEW CONTROLLER (MVC)

What is Spring MVC framework?

Spring comes with a full-featured MVC framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring's MVC framework uses IoC to provide a clean separation of controller logic from business objects. It also allows to declaratively bind request parameters to business objects.

DispatcherServlet

The Spring Web MVC framework is designed around a `DispatcherServlet` that handles all the HTTP requests and responses.

WebApplicationContext

The `WebApplicationContext` is an extension of the plain `ApplicationContext` that has some extra features necessary for web applications. It differs from a normal `ApplicationContext` in that it is capable of resolving themes, and that it knows which servlet it is associated with.

What is Controller in Spring MVC framework?

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

@Controller annotation

The `@Controller` annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

@RequestMapping annotation

`@RequestMapping` annotation is used to map a URL to either an entire class or a particular handler method.