1. **What is Spring?**

It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in java. Spring framework targets to make Java EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

1. **What are features of Spring?**

* **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 using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.
* **Aspect oriented (AOP)** − Spring supports Aspect oriented programming and enables cohesive development by separating 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 such as Struts or other over engineered or less popular web frameworks.
* **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).
* **Exception Handling** − Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

1. **What are the different modules in Spring framework?**

* Core module
* Bean module
* Context module
* Expression Language module
* JDBC module
* ORM module
* OXM module
* Java Messaging Service(JMS) module
* Transaction module
* Web module
* Web-Servlet module
* Web-Struts module
* Web-Portlet module

1. **What is Dependency Injection in Spring?**

Dependency Injection, an aspect of Inversion of Control (IoC), is a design pattern to provide loose coupling. It removes the dependency from the program.

|  |  |
| --- | --- |
| 1. **Without IOC & DI** | 1. **With IOC & DI** |
| 1. **public** **class** Employee{ 2. Address address; 3. Employee(){ 4. address=**new** Address();//creating instance 5. } 6. } | 1. **public** **class** Employee{ 2. Address address; 3. Employee(Address address){ 4. **this**.address=address;//not creating instance 5. } 6. } |
| 1. Now, there is dependency between Employee and Address because Employee is forced to use the same address instance. | 1. Now, there is no dependency between Employee and Address because Employee is not forced to use the same address instance. It can use any address instance. |

1. **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.

The best solution is using 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.

1. **What are the benefits of IOC?**

* It minimizes the amount of code in your application.
* It makes your application easy to test as it doesn't require any singletons or JNDI lookup mechanisms in your unit test cases.
* Loose coupling is promoted with minimal effort and least intrusive mechanism.
* IOC containers support eager instantiation and lazy loading of services.

1. **What is the role of IOC container in spring?**

* create the instance
* configure the instance, and
* assemble the dependencies

1. **What are the types of IOC container in spring?**

There are two types of IOC containers in spring framework.

* **Bean Factory container** − This is the simplest container providing basic support for DI .The BeanFactory is usually preferred where the resources are limited like mobile devices or applet based applications
* **Spring ApplicationContext Container** – This extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc.

1. **Give an example of BeanFactory implementation?**

The most commonly used BeanFactory implementation is the XmlBeanFactory class. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

1. **What are the common implementations of the ApplicationContext?**

The three commonly used implementation of 'Application Context' are −

* **FileSystemXmlApplicationContext** − This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.
* **ClassPathXmlApplicationContext** − This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.
* **WebXmlApplicationContext** − This container loads the XML file with definitions of all beans from within a web application.

1. **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 <bean/> 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.

1. **What does a bean definition contain?**

The bean definition contains the information called configuration metadata which is needed for the container to know the followings −

* How to create a bean
* Bean's lifecycle details
* Bean's dependencies

1. **How do you provide configuration metadata to the Spring Container?**

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

1. **How do you define the scope of a bean?**

When defining a <**bean**> 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**.

1. **Explain the bean scopes supported by Spring?**

* **singleton:** Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues **because it’s not thread-safe**.
* **prototype**: A new instance will be created every time the bean is requested.
* **request**: This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
* **session**: A new bean will be created for each HTTP session by the container.
* **global-session:** This is used to create global session beans for Portlet applications.

1. **Explain Bean lifecycle in Spring framework?**

* **Instantiate:** First the spring container finds the bean’s definition from the XML file and instantiates the bean.
* **Populate properties:** Using the dependency injection, spring populates all of the properties as specified in the bean definition.
* **Set Bean Name:** If the bean implements BeanNameAware interface, spring passes the bean’s id to setBeanName() method.
* **Set Bean factory:** If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
* **Pre Initialization:** Also called post process of bean. If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.
* **Initialize beans:** If the bean implements IntializingBean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
* **Post Initialization:**– If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
* **Ready to use:** Now the bean is ready to use by the application
* **Destroy:** If the bean implements DisposableBean , it will call the destroy() method

1. **What are inner beans in Spring?**

When a bean is only used as a property of another bean it can be declared as an inner bean. Spring’s XML-based configuration metadata provides the use of <bean/> element inside the <property/> or <constructor-arg/> elements of a bean definition, in order to define the so-called inner bean. Inner beans are always anonymous and they are always scoped as prototypes.

1. **How can you inject Java Collection in Spring?**

* **<list>** − This helps in wiring i.e. injecting a list of values, allowing duplicates.
* **<set>** − This helps in wiring a set of values but without any duplicates.
* **<map>** − This can be used to inject a collection of name-value pairs where name and value can be of any type.
* **<props>** − This can be used to inject a collection of name-value pairs where the name and value are both Strings.

1. **What is autowiring in spring? What are the autowiring modes?**

Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic. Let's see the code to inject bean using dependency injection.

<bean id="emp" **class**="com.javatpoint.Employee" autowire="byName" />

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | this is the default mode, it means autowiring is not enabled. |
| 2) | byName | injects the bean based on the property name. It uses setter method. |
| 3) | byType | injects the bean based on the property type. It uses setter method. |
| 4) | constructor | It injects the bean using constructor |
| 5) | Autodetect | Deprecated since Spring 3.  Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType. |

1. **What are the limitations with autowiring?**

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

1. **Can you inject null and empty string values in Spring?**

Yes.

1. **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.

1. **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 itself by using annotations on the relevant class, method, or field declaration.

1. **How do you turn on annotation wiring?**

Annotation wiring is not turned on in the Spring container by default. In order to use annotation based wiring we must enable it in our Spring configuration file by configuring **<context:annotation-config/>** element.

1. **What does @Required annotation mean?**

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.

1. **What does @Autowired annotation mean?**

This 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.

1. **What does @Qualifier annotation mean?**

There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property, in such case you can use **@Qualifier** annotation along with **@Autowired** to remove the confusion by specifying which exact bean will be wired.

1. **How is event handling done in Spring?**

Event handling in the **ApplicationContext** is provided through the **ApplicationEvent** **class** and **ApplicationListener** **interface**. So if a bean implements the **ApplicationListener**, then every time an **ApplicationEvent** gets published to the **ApplicationContext**, that bean is notified.

1. **Describe some of the standard Spring events?**

* **ContextRefreshedEvent** − This event is published when the ApplicationContext is either initialized or refreshed. This can also be raised using the refresh() method on the ConfigurableApplicationContext interface.
* **ContextStartedEvent** − This event is published when the ApplicationContext is started using the start() method on the ConfigurableApplicationContext interface. You can poll your database or you can re/start any stopped application after receiving this event.
* **ContextStoppedEvent** − This event is published when the ApplicationContext is stopped using the stop() method on the ConfigurableApplicationContext interface. You can do required housekeep work after receiving this event.
* **ContextClosedEvent** − This event is published when the ApplicationContext is closed using the close() method on the ConfigurableApplicationContext interface. A closed context reaches its end of life; it cannot be refreshed or restarted.
* **RequestHandledEvent** − This is a web-specific event telling all beans that an HTTP request has been serviced.

1. **What is Aspect?**

A module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the **@Aspect** annotation (@AspectJ style).

1. **What is the difference between concern and cross-cutting concern in Spring AOP?**

**Concern** − Concern is behavior which we want to have in a module of an application. Concern may be defined as a functionality we want to implement. Issues in which we are interested define our concerns.

**Cross-cutting concern** − It's a concern which is applicable throughout the application and it affects the entire application. e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence are cross-cutting concerns.

1. **What is Join point?**

This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.

JoinPoint is any point in your program such as field access, method execution, exception handling etc.

1. **What is Advice?**

This is the actual action to be taken either before or after the method execution. This is actual piece of code that is invoked during program execution by Spring AOP framework.

Advice represents action taken by aspect.

1. **What are the types of advice?**

* **before** − Run advice before the a method execution.
* **after** − Run advice after the a method execution regardless of its outcome.
* **after-returning** − Run advice after the a method execution only if method completes successfully.
* **after-throwing** − Run advice after the a method execution only if method exits by throwing an exception.
* **around** − Run advice before and after the advised method is invoked.

1. **What is Pointcut?**

This is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.

Pointcut is expression language of Spring AOP.

1. **What is Introduction?**

An introduction allows you to add new methods or attributes to existing classes.

Introduction represents introduction of new fields and methods for a type.

1. **What is target object?**

The object being advised by one or more aspects, this object will always be a proxy object. Also referred to as the advised object.

1. **What is interceptor?**

Interceptor is a class like aspect that contains one advice only.

1. **What is weaving? What are the different points where weaving can be applied?**

Weaving is a process of linking aspect with other application. Weaving can be done at compile time, load time, or at runtime. Spring framework performs weaving at runtime.

1. **What are the AOP implementation?**
2. Spring AOP
3. Apache AspectJ
4. JBoss AOP
5. **What is @AspectJ? based aspect implementation?**

@**AspectJ** refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

1. **What is Spring MVC framework?**

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

1. **What is a DispatcherServlet?**

The Spring Web MVC framework is designed around a **DispatcherServlet** that handles all the HTTP requests and responses. The **DispatcherServlet** class works as the **front controller** in Spring MVC.

1. **What does @Controller annotation?**

The **@Controller** annotation marks the class as controller class. It is applied on the class. Spring does not require you to extend any controller base class or reference the Servlet API.

1. **Explain @RequestMapping annotation?**

The **@RequestMapping** annotation maps the request with the method. It is applied on the method. **@RequestMapping** annotation is used to map a URL to either an entire class or a particular handler method.

1. **What does the ViewResolver class?**

The **View Resolver** class resolves the view component to be invoked for the request. It defines prefix and suffix properties to resolve the view component.

1. **Which ViewResolver class is widely used?**

The **org.springframework.web.servlet.view.InternalResourceViewResolver** class is widely used.

1. **Does spring MVC provide validation support?**

Yes.

1. **What is 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.

1. **What are the ways to access Hibernate by using Spring?**

There are two ways to access hibernate using spring −

* Inversion of Control with a Hibernate Template and Callback.
* Extending **HibernateDAOSupport** and Applying an AOP Interceptor node.

1. **What are the types of the transaction management Spring supports?**

Spring supports two types of transaction management −

* **Programmatic transaction management** − This means that you have managed the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.
* **Declarative transaction management** − This means you separate transaction management from the business code. You only use annotations or XML based configuration to manage the transactions.

**Declarative transaction management** is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code.

1. **What are the advantages of JdbcTemplate in spring?**

**Less code**: By using the JdbcTemplate class, you don't need to create connection,statement,start transaction,commit transaction and close connection to execute different queries. You can execute the query directly.

1. **What are classes for spring JDBC API?**

* JdbcTemplate
* SimpleJdbcTemplate
* NamedParameterJdbcTemplate
* SimpleJdbcInsert
* SimpleJdbcCall

1. **How can you fetch records by spring JdbcTemplate?**

You can fetch records from the database by the query method of JdbcTemplate. There are two interfaces to do this:

* ResultSetExtractor
* RowMapper

1. **What is the advantage of NamedParameterJdbcTemplate?**

**NamedParameterJdbcTemplate** class is used to pass value to the named parameter. A named parameter is better than ? (question mark of **PreparedStatement**).

1. **What is the advantage of SimpleJdbcTemplate?**

The **SimpleJdbcTemplate** supports the feature of **var-args** and **autoboxing**.