**ASSIGNMENT-SPRING CORE**

1.Why use spring framework?

**Ans:**

Spring improves coding efficiency and reduces overall application development time because

* it is lightweight.
* efficient at utilizing system resources.
* and helps in loose coupling.

Spring removes tedious configurxation work so that developers can focus on writing business logic.

2. Dependency:

Ans:

A simple application will have more than one objects. When an object works with another object then it is called dependency. When an object having reference to another object for functionality, it is referred as dependent object.

Example:

When preparing bill, the billing class requires the object of customer details and product details class, these objects are injected in the billing class.

3.Dependency Injection:

Ans:

Dependency injection is a design pattern that removes dependency of a program in such way that we provide information from an external file such as xml file or a java file.

It makes the code loosely coupled and easier for testing.

The external source helps to inject one object to another.

4. Inversion of Control:

Ans:

 It means giving the control of creating and instantiating the spring beans to the Spring IOC container and the only work the developer does is configuring the beans in the spring xml file.

It is a process of inverting a control to xml entity

In applications we invert to control for two things

* Creation of object
* Dependency Injection

5.IOC Container:

Ans:

IoC Container **manages java objects**

* from instantiation to destruction through its Bean Factory.
* Java components(class) that are instantiated by the IoC container are called beans.
* **IoC container manages a bean's scope, lifecycle events** for which it has been configured and coded.

6.ApplicationContext

Ans:

The Application Context is Spring's advanced container.

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

Commonly used ApplicationContext implementations are

* **FileSystemXmlApplicationContext**
* **ClassPathXmlApplicationContext**
* **WebXmlApplicationContext**

7.BeanFactory:

Ans:

BeanFactory interface is the simplest container providing an advanced configuration mechanism to instantiate, configure and manage the life cycle of beans.

BeanFactory represents a basic IoC container which is a parent interface of ApplicationContext.

BeanFactory loads the bean definitions and dependency amongst the beans based on a configuration file(XML) or the beans can be directly returned when required using Java Configuration.

8.AbstractApplicationContext:

Ans:

It is an abstract class implementing ApplicaionConfigurableContext.

It simply implements common functionality ie) Abstract implementation of Application context.

9.Configuration Metadata

Ans:

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

The configuration metadata is represented in XML, Java annotations, or Java code. It allows you to express the objects that compose your application and the interdependencies between objects.

10.Bean:

Ans:

A bean is an object that is instantiated, assembled, and managed by a Spring IoC container.

These beans are created with the configuration metadata that you supply to the container.

11.Constructor Injection:

Ans:

Injecting dependency through constructor is called constructor injection.

It is done by using <constructor-arg>, which is placed inside the <bean>.

12.Setter Injection:

Ans:

Setter injection is a dependency injection in which the spring framework injects the dependency object using the setter method.

It is done through the bean-configuration file by <property> tag.

13. **Autowiring:**

**Ans:**

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference only.

**14.Modes of Autowiring:**

* no - it is the default autowiring mode. It means no autowiring bydefault.
* byName- The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
* byType - The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.
* constructor - The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.

15. Bean Scopes:

Ans:

Scope defines the lifetime of the bean.

There are five types of bean scope. They are

* Singleton.
* Prototype.
* Session.
* Request.
* Global Session.

16.Singleton:

Ans:

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.

17.Prototype:

Ans:

If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made. As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.

18.Limitations of autowiring:

Ans:

* Overriding possibilities.
* Cannot be used in primitive datatypes.
* Confusing nature.

19.Spring Bean life cycle:

Ans:

A Spring bean needs to be instantiated when the container starts, based on Java or XML bean definition. The framework may also be required to perform some pre and post-initialization steps to get the bean into a usable state.

After that, when the bean is no longer required, it will be removed from the IoC container. Like the initialization phase, the Spring framework may need to perform pre-and post-destruction steps to free the other system resources.

Cycle:

* Initialization.
* Bean Usage.
* Destruction.

It can be done in three ways:

Lifecycle callback hooks

* Initializing Bean
* Disposable Bean

JSK-250 Annotation

* @PostConstruct
* @PreDestroy

By init() and destroy() method.

20.Annotations in spring.

Ans:

@Autowired annotation is applied to the fields, setter methods, and constructors. It injects object dependency implicitly.

The @Qualifier annotation is used to resolve the autowiring conflict when there are multiple beans of the same type.

@Primary indicates that a particular bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency. If exactly one ‘primary’ bean exists among the candidates, it will be the autowired value.

@Bean A method-level annotation used to declare a spring bean.

@Component-Indicates that an annotated class is a “component” and will be auto-detected when using annotation-based configuration and classpath scanning.

@Scope-Indicates the name of a scope to use for instances of the annotated type.

@Lookup-Indicates a method as ‘lookup’ method. It is best used for injecting a prototype-scoped bean into a singleton bean.

@DependsOn-During component-scanning, it is used to specify the beans on which the current bean depends on. The specified beans are guaranteed to be created by the container before this bean.

@Lazy-Indicates whether a bean is to be lazily initialized. By default, in spring DI, eager initialization will occur.

When applied over any bean, initialization of that bean will not happen until referenced by another bean or explicitly retrieved from the enclosing BeanFactory.

@Value-Applicable at the field or method/constructor parameter level, and indicates a default value expression for the affected argument.

@Required-Default bean autowiring checks only that dependency has been set. It does not checks whether the assigned value is null or not. Using @Required, we can check if values set are non-null. It has been deprecated now.

@Controller- A specialization of the @Component to annotate controllers (e.g. a web controller). It used in combination with annotated handler methods based on the RequestMapping annotation.

@Service -A specialization of the @Component annotation. It indicates that a class is a “Business Service Facade” or something similar.

@Repository- A specialization of the @Component annotation. In addition to importing the annotated DAO classes into the DI container, it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation into Spring DataAccessException

@Configuration-Placed at the top of the java configuration class.

BY

R LAVANYA .