Spring Interview Questions

1) What is Spring?

Spring is a lightweight inversion of control and aspect-oriented container framework.

2) Explain Spring?

- **Lightweight : *Spring*** is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB. And the processing overhead is also very negligible.
- **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.
- **Framework : *Spring*** provides most of the intra functionality leaving rest of the coding to the developer.

3) What are the different modules in Spring framework?

- The Core container module
- Application context module
- AOP module (Aspect Oriented Programming)
- JDBC abstraction and DAO module
- O/R mapping integration module (Object/Relational)
- Web module
- ***MVC framework*** module

4) What is the structure of Spring framework?

5) What is the Core container module?

This module is provides the fundamental functionality of the spring framework. In this module **BeanFactory** is the heart of any spring-based application. The entire framework was built on the top of this module. This module makes the ***Spring container***.

6) What is Application context module?

The Application context module makes spring a framework. This module extends the concept of ***BeanFactory***, providing support for internationalization (I18N) messages, application lifecycle events, and validation. This module also supplies many enterprise

services such JNDI access, ***EJB integration***, remoting, and scheduling. It also provides support to other framework.

7) What is AOP module?

The ***AOP*** module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between ***Spring*** and other ***AOP*** frameworks. This module also introduces metadata programming to ***Spring***. Using Spring's metadata support, we will be able to add ***annotations*** to our source code that instruct ***Spring*** on where and how to apply aspects.

8) What is JDBC abstraction and DAO module?

Using this module we can keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. A new layer of meaningful exceptions on top of the error messages given by several database servers is bought in this module. In addition, this module uses ***Spring's AOP module*** to provide transaction management services for objects in a Spring application.

9) What are 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 provide 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***.

10) What is web module?

This 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***.

11) What is web module?

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 for a clean separation of controller logic from business objects. It also allows you to declaratively bind request parameters to your business objects. It also can take advantage of any of Spring's other services, such as I18N messaging and validation.

12) What is a BeanFactory?

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.

13) What is AOP Alliance?

AOP Alliance is an open-source project whose goal is to promote adoption of AOP and interoperability among different AOP implementations by defining a common set of interfaces and components.

14) What is Spring configuration file?

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

15) What does a simple spring application contain?

These applications are like any Java application. They are made up of several classes, each performing a specific purpose within the application. But these classes are configured and introduced to each other through an XML file. This XML file describes how to configure the classes, known as the Spring configuration file.

16) What is XMLBeanFactory?

BeanFactory has many implementations in Spring. But one of the most useful one is ***org.springframework.beans.factory.xml.XmlBeanFactory***, which loads its beans based on the definitions contained in an XML file. To create an ***XmlBeanFactory***, pass a java.io.InputStream to the constructor. The ***InputStream*** will provide the XML to the factory. For example, the following code snippet uses a java.io.***FileInputStream*** to provide a bean definition XML file to ***XmlBeanFactory***.

```
BeanFactory factory = new XmlBeanFactory(new FileInputStream("beans.xml"));
```

To retrieve the bean from a BeanFactory, call the getBean() method by passing the name of the bean you want to retrieve.

```
MyBean myBean = (MyBean) factory.getBean("myBean");
```

17) What are important ApplicationContext implementations in spring framework?

- **ClassPathXmlApplicationContext –** This context loads a context definition from an XML file located in the class path, treating context definition files as class path resources.
- **FileSystemXmlApplicationContext –** This context loads a context definition from an XML file in the filesystem.

- **XmlWebApplicationContext –** This context loads the context definitions from an XML file contained within a web application.

18) Explain Bean lifecycle in Spring framework?

1. The spring container finds the bean's definition from the XML file and instantiates the bean.
2. Using the dependency injection, spring populates all of the properties as specified in the bean definition.
3. If the bean implements the **BeanNameAware** interface, the factory calls **setBeanName()** passing the bean's ID.
4. If the bean implements the **BeanFactoryAware** interface, the factory calls **setBeanFactory()**, passing an instance of itself.
5. If there are any **BeanPostProcessors** associated with the bean, their **post-ProcessBeforeInitialization()** methods will be called.
6. If an init-method is specified for the bean, it will be called.
7. Finally, if there are any **BeanPostProcessors** associated with the bean, their **postProcessAfterInitialization()** methods will be called.

19) What is bean wiring?

Combining together beans within the Spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together.

20) How do add a bean in spring application?

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/d
<beans>
        <bean id="foo" class="com.act.Foo"/>
        <bean id="bar" class="com.act.Bar"/>
</beans>
```

In the bean tag the id attribute specifies the bean name and the class attribute specifies the fully qualified class name.

21) What are singleton beans and how can you create prototype beans?

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.

```
<beans>
   <bean id="bar" class="com.act.Foo" singleton="false"/>
</beans>
```

22) What are the important beans lifecycle methods?

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.

23) How can you override beans default lifecycle methods?

The bean tag has two more important attributes with which you can define your own custom initialization and destroy methods. Here I have shown a small demonstration. Two new methods fooSetup and fooTeardown are to be added to your Foo class.

```
<beans>
  <bean id="bar" class="com.act.Foo" init-method=”fooSetup” destroy=”fooTeardown”/
</beans>
```

24) What are Inner Beans?

When wiring beans, if a bean element is embedded to a property tag directly, then that bean is said to the Inner Bean. The drawback of this bean is that it cannot be reused anywhere else.

25) What are the different types of bean injections?

There are two types of bean injections.

1. **By setter**
2. **By constructor**

26) What is Auto wiring?

You can wire the beans as you wish. But spring framework also does this work for you. It can auto wire the related beans together. All you have to do is just set the autowire attribute of bean tag to an autowire type.

```
<beans>
  <bean id="bar" class="com.act.Foo" Autowire=”autowire type”/>
</beans>
```

27) What are different types of Autowire types?

There are four different types by which autowiring can be done.

- **byName**
- **byType**
- **constructor**
- **autodetect**

28) What are the different types of events related to Listeners?

There are a lot of events related to **ApplicationContext** of spring framework. All the events are subclasses of **org.springframework.context.Application-Event**. They are

- ContextClosedEvent – This is fired when the context is closed.
- ContextRefreshedEvent – This is fired when the context is initialized or refreshed.
- RequestHandledEvent – This is fired when the web context handles any request.

29) What is an Aspect?

An aspect is the cross-cutting functionality that you are implementing. It is the aspect of your application you are modularizing. An example of an aspect is logging. Logging is something that is required throughout an application. However, because applications tend to be broken down into layers based on functionality, reusing a logging module through inheritance does not make sense. However, you can create a logging aspect and apply it throughout your application using AOP.

30) What is a Jointpoint?

A joinpoint is a point in the execution of the application where an aspect can be plugged in. This point could be a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect's code can be inserted into the normal flow of your application to add new behavior.

31) What is an Advice?

Advice is the implementation of an aspect. It is something like telling your application of a new behavior. Generally, and advice is inserted into an application at joinpoints.

32) What is a Pointcut?

A pointcut is something that defines at what joinpoints an advice should be applied. Advices can be applied at any joinpoint that is supported by the AOP framework. These Pointcuts allow you to specify where the advice can be applied.

33) What is an Introduction in AOP?

An introduction allows the user to add new methods or attributes to an existing class. This can then be introduced to an existing class without having to change the structure of the class, but give them the new behavior and state.

34) What is a Target?

A target is the class that is being advised. The class can be a third party class or your own class to which you want to add your own custom behavior. By using the concepts of AOP,

the target class is free to center on its major concern, unaware to any advice that is being applied.

35) What is a Proxy?

A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

36) What is meant by Weaving?

The process of applying aspects to a target object to create a new proxy object is called as Weaving. The aspects are woven into the target object at the specified joinpoints.

37) What are the different points where weaving can be applied?

- Compile Time
- Classload Time
- Runtime

38) What are the different advice types in spring?

- **Around :** Intercepts the calls to the target method
- **Before :** This is called before the target method is invoked
- **After :** This is called after the target method is returned
- **Throws :** This is called when the target method throws and exception
- Around : org.aopalliance.intercept.MethodInterceptor
- Before : org.springframework.aop.BeforeAdvice
- After : org.springframework.aop.AfterReturningAdvice
- Throws : org.springframework.aop.ThrowsAdvice

39) What are the different types of AutoProxying?

- BeanNameAutoProxyCreator
- DefaultAdvisorAutoProxyCreator
- Metadata autoproxying

40) What is the Exception class related to all the exceptions that are thrown in spring applications?

DataAccessException - org.springframework.dao.DataAccessException

41) What kind of exceptions those spring DAO classes throw?

The springâ€™s DAO class does not throw any technology related exceptions such as SQLException. They throw exceptions which are subclasses of DataAccessException.

42) What is DataAccessException?

DataAccessException is a RuntimeException. This is an Unchecked Exception. The user is not forced to handle these kinds of exceptions.

43) How can you configure a bean to get DataSource from JNDI?

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
     <property name="jndiName">
       <value>java:comp/env/jdbc/myDatasource</value>
    </property>
</bean>
```

44) How can you create a DataSource connection pool?

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
   <property name="driver">
       <value>${db.driver}</value>
          </property>
          <property name="url">
      <value>${db.url}</value>
          </property>
          <property name="username">
     <value>${db.username}</value>
          </property>
          <property name="password">
    <value>${db.password}</value>
          </property>
</bean>
```

45) How JDBC can be used more efficiently in spring framework?

JDBC can be used more efficiently with the help of a template class provided by spring framework called as **JdbcTemplate**.

46) How JdbcTemplate can be used?

With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database.

```
JdbcTemplate template = new JdbcTemplate(myDataSource);
```

A simple DAO class looks like this.

```
public class StudentDaoJdbc implements StudentDao {
  private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
   this.jdbcTemplate = jdbcTemplate;
}
more..
}
```

The configuration is shown below.

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
   <property name="dataSource">
       <ref bean="dataSource"/>
   </property>
</bean>
<bean id="studentDao" class="StudentDaoJdbc">
   <property name="jdbcTemplate">
       <ref bean="jdbcTemplate"/>
   </property>
</bean>
<bean id="courseDao" class="CourseDaoJdbc">
   <property name="jdbcTemplate">
       <ref bean="jdbcTemplate"/>
   </property>
</bean>
```

47) How do you write data to backend in spring using JdbcTemplate?

The JdbcTemplate uses several of these callbacks when writing data to the database. The usefulness you will find in each of these interfaces will vary. There are two simple interfaces. One is **PreparedStatementCreator** and the other interface is **BatchPreparedStatementSetter**.

48) Explain about PreparedStatementCreator?

PreparedStatementCreator is one of the most common used interfaces for writing data to database. The interface has one method createPreparedStatement().

```
PreparedStatement createPreparedStatement(Connection conn)
throws SQLException;
```

When this interface is implemented, we should create and return a PreparedStatement from the Connection argument, and the exception handling is automatically taken care off. When this interface is implemented, another interface **SqlProvider** is also implemented which has a method called **getSql()** which is used to provide sql strings to JdbcTemplate.

49) Explain about BatchPreparedStatementSetter?

If the user what to update more than one row at a shot then he can go for **BatchPreparedStatementSetter**. This interface provides two methods

```
setValues(PreparedStatement ps, int i) throws SQLException;

int getBatchSize();
```

The getBatchSize() tells the JdbcTemplate class how many statements to create. And this also determines how many times setValues() will be called.

50) Explain about RowCallbackHandler and why it is used?

In order to navigate through the records we generally go for ResultSet. But spring provides an interface that handles this entire burden and leaves the user to decide what to do with each row. The interface provided by spring is **RowCallbackHandler**. There is a method processRow() which needs to be implemented so that it is applicable for each and everyrow.

```
void processRow(java.sql.ResultSet rs);
```