================INTERVIEW QUESTIONS=============

## 1) What is Spring Framework?

***Spring*** is a lightweight inversion of control and aspect-oriented container framework. Spring Framework’s contribution towards java community is immense and spring community is the largest and most innovative community by size. They have numerous projects under their portfolio and have their own [spring dmServer](http://www.springsource.com/products/dmserver) for running spring applications. This community is acquired by VMWare, a leading cloud compting company for enabling the java application in the cloud by using spring stacks. If you are looking to read more about the spring framework and its products, please read in their official site [Spring Source](http://www.springsource.com/).

**IOC and DI:**

* Spring helps in creating loosely coupled application because of **Dependency Injection**.
* In spring objects define their associations (dependencies) and do not worry about how to get those **dependencies** ; now it is the responsibility of Spring to provide the required dependencies for creating objects.

**For example** : Suppose we have an object Employee and it has a dependency on object Address. So we define a bean corresponding to Employee where it will define its dependency on object Address. When Spring tries to create an Object Employee it sees that Employee has a dependency on object Address so first it will create the Address object (dependent object) and then inject this into the Employee Object.

* Inversion of Control **(IOC)** and Dependency Injection **(DI)** are used interchangeably. IOC is achieved through DI. DI is the process of providing the dependencies and IOC is the end result of DI (**Note:** DI is not the only way to achieve IOC, there are [other ways](https://en.wikipedia.org/wiki/Inversion_of_control#Implementation_techniques) as well).
* By DI the responsibility of creating objects is shifted from our application code to Spring container hence the phenomenon is called IOC.
* Dependency Injection can be done by setter injection, constructor injection (2 Examples).

|  |  |
| --- | --- |
| ApplicationContext.xml | Java Class |
| 1. <bean id="e" **class**="com.javatpoint.Employee"> 2. <constructor-arg value="10" type="int"></constructor-arg> 3. </bean> | 1. **public** **class** Employee { 2. **private** **int** id; 3. **private** String name;   **public** Employee() {System.out.println("def cons");}   1. **public** Employee(**int** id) {**this**.id = id;}   **public** Employee(String name) {  **this**.name = name;}  } |
| <bean id = "textEditor" class = "com.tutorialspoint.TextEditor">  <constructor-arg ref = "spellChecker"/>  </bean>  <bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></bean> | public class TextEditor {  private SpellChecker spellChecker;  public TextEditor(SpellChecker spellChecker) {  this.spellChecker = spellChecker;}  public void spellCheck() {  spellChecker.checkSpelling();  }  } |

# [Explain why constructor inject is better than other options](https://stackoverflow.com/questions/21218868/explain-why-constructor-inject-is-better-than-other-options):

|  |  |  |
| --- | --- | --- |
| **Setter Injection** |  | **Constructor Injection** |
| **1.** In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives |  | **1.** In constructor injection, partial injection of dependencies cannot possible, because for calling constructor we must pass all the arguments right, if not so we may get error |
| **2.**Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property [i already told regarding this, hope you remember ] |  | **2.** But, constructor injection cannot overrides the setter injected values |
| **3.** If we have more dependencies for example 15 to 20 are there in our bean class then, in this case setter injection is not recommended as we need to write almost 20 setters right, bean length will increase. |  | **3.** In this case, Constructor injection is highly recommended, as we can inject all the dependencies with in 3 to 4 lines [i mean, by calling one constructor] |
| **4.** Setter injection makes bean class object as mutable [We can change ] |  | **4.** Constructor injection makes bean class object as immutable [We cannot change ] |

**Popular Tutorials**

* ***[Spring Tutorial](https://www.dineshonjava.com/2012/06/spring-30-baby-step-to-learn.html)***
* **[Spring MVC Web Tutorial](https://www.dineshonjava.com/2012/12/spring-web-mvc-framework-chapter-38.html)**
* ***[Spring Boot Tutorial](https://www.dineshonjava.com/2016/06/introduction-to-spring-boot-a-spring-boot-complete-guide.html)***
* **[Spring Security Tutorial](https://www.dineshonjava.com/2013/02/spring-security-take-baby-step-to-secure.html)**
* **[Spring AOP Tutorial](https://www.dineshonjava.com/2017/03/spring-aop-tutorial-with-example-aspect-advice-pointcut-joinpoint.html)**
* **[Spring JDBC Tutorial](https://www.dineshonjava.com/2012/12/using-spring-jdbc-framework-chapter-32.html)**
* ***[Spring HATEOAS](https://www.dineshonjava.com/2017/01/spring-hateoas-hypermedia-driven-restful-web-service.html)***
* ***[Microservices with Spring Boot](https://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html)***
* **[REST Webservice](https://www.dineshonjava.com/2013/06/jax-rs-web-service-tutorial.html)**
* ***[Core Java](https://www.dineshonjava.com/2013/01/core-java-baby-step-to-be-best-java-ian.html)***
* ***[Hibernate Tutorial](https://www.dineshonjava.com/2012/03/hibernate-3-on-baby-steps.html)***
* **[Spring Batch](https://www.dineshonjava.com/2012/12/spring-batch-process-with-example.html)**

## 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 ------------ This module makes the ***Spring container***.
* 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.



Message I18n:

<!-- START : Properties File Reader -->

<bean id=*"messageSource"*

class=*"org.springframework.context.support.ResourceBundleMessageSource"*>

<property name=*"basenames"*>

<list>

<value>properties.customerprofile</value>

<value>properties.event\_types\_and\_code\_mapping</value>

</list>

</property>

</bean>

<!-- END : Properties File Reader -->

## 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.

|  |
| --- |
| <!-- START : Transaction manager and AOP configurations -->  <bean id=*"transactionManager"* class=*"org.springframework.orm.jpa.JpaTransactionManager"*>  <property name=*"entityManagerFactory"* ref=*"entityManagerFactory"* />  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- CP Manager Transactions -->  <aop:config>  <aop:pointcut id=*"customerProfileManagerPointCut"*  expression=*"execution(\* \*..CustomerProfileManager.\*(..))"* />  <aop:advisor advice-ref=*"cpManagerTransactionAdvice"*  pointcut-ref=*"customerProfileManagerPointCut"* />  </aop:config>  <tx:advice id=*"cpManagerTransactionAdvice"*  transaction-manager=*"transactionManager"*>  <tx:attributes>  <tx:method name=*"createCustomerProfile"* propagation=*"REQUIRED"* />  <tx:method name=*"updateCustomerProfile"* propagation=*"REQUIRED"*  rollback-for=*"Exception"* />  <tx:method name=*"retrieveCustomerProfile"* propagation=*"REQUIRED"* />  <tx:method name=*"getCustomerEvents"* propagation=*"REQUIRED"* />  </tx:attributes>  </tx:advice> |

## 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.

<!-- START : DataSource, JMS Connection Factory and JNDI Beans -->

<bean id=*"dataSource"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"resourceRef"*>

<value>false</value>

</property>

<property name=*"jndiName"*>

<value>${customerprofile.datasource}</value>

</property>

</bean>

## 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. Spring’s transaction management supports each of these ***ORM frameworks*** as well as ***JDBC***.

|  |
| --- |
| <!-- START : Persistence Manager beans -->  <bean class=*"org.springframework.orm.jpa.support.*  *PersistenceAnnotationBeanPostProcessor"* />  <bean id=*"persistenceManager"* class=*"aero.sita.csp.esb2.persistence.PersistenceManagerImpl"*>  <property name=*"entityManagerFactory"* ref=*"entityManagerFactory"* />  </bean>  <bean id=*"entityManagerFactory"*  class=*"org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"* />  <property name=*"persistenceUnitName"* value=*"CustomerProfile"* />  <property name=*"persistenceUnitManager"* ref=*"persistenceUnitManager"* />  <!--You need to tell spring who the persistence provider is so that it can be used to create the EntityManagerFactory    And persist both JPA and JDBC operations for that jpaVendorAdapter used- >  <property name=*"jpaVendorAdapter"*>  //inner bean  <bean id=*"jpaVendorAdapter"* class=*"org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"*>  <property name=*"database"* value=*"ORACLE"* />  <property name=*"showSql"* value=*"false"* />  </bean>  </property>  <property name=*"loadTimeWeaver"*>  <bean  class=*"org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"* />  </property>  </bean>    <bean id=*"persistenceUnitManager"* class=*"aero.sita.csp.esb2.persistence.CSPPersistenceManager"*>  <property name=*"persistenceXmlLocations"*>  <list>  <value>classpath\*:META-INF/persistence.xml</value>  </list>  </property>  <property name=*"defaultDataSource"* ref=*"dataSource"* />  </bean>  <!-- END : Persistence Manager beans --> |
| Persistant.xml  <?xml version=*"1.0"*?>  <persistence xmlns=*"http://java.sun.com/xml/ns/persistence"*  version=*"1.0"*>  <persistence-unit name=*"CustomerProfile"*  transaction-type=*"RESOURCE\_LOCAL"*>  <provider>org.hibernate.ejb.HibernatePersistence</provider>  <properties>  <property name=*"hibernate.dialect"* value=*"org.hibernate.dialect.OracleDialect"* />    <property name=*"hibernate.default\_batch\_fetch\_size"* value=*"25"* />  <property name=*"hibernate.jdbc.fetch\_size"* value=*"25"* />  <property name=*"hibernate.jdbc.batch\_size"* value=*"25"* />  <property name=*"hibernate.order\_inserts"* value=*"true"*/>  <property name=*"hibernate.order\_updates"* value=*"true"*/>  </properties>  </persistence-unit>  </persistence> |
|  |

## 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 decoratively 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***.

|  |  |
| --- | --- |
| 1  2 | 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.

|  |  |
| --- | --- |
| 1 | MyBean myBean = (MyBean) factory.getBean(“myBean”); |

## 17) What are important ApplicationContext implementations in spring framework? FCX

* **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?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?xml version=”1.0″ encoding=”UTF-8″?>    <!DOCTYPE beans PUBLIC “-//SPRING//DTD BEAN//EN”              “http://www.springframework.org/dtd/spring-beans.dtd”>  <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.

|  |  |
| --- | --- |
| 1  2  3  4 | <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.

|  |  |
| --- | --- |
| 1  2  3  4 | <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.

|  |  |
| --- | --- |
| 1  2  3 | <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?

|  |  |
| --- | --- |
| 1  2 | DataAccessException –     org.springframework.dao.DataAccessException |

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

The spring’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?

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <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?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <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.

|  |  |
| --- | --- |
| 1 | <strong>JdbcTemplate</strong> template = new <strong>JdbcTemplate</strong>(myDataSource); |

A simple DAO class looks like this.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class StudentDaoJdbc implements StudentDao {      private JdbcTemplate jdbcTemplate;      public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {         this.jdbcTemplate = jdbcTemplate;      } more..  } |

The configuration is shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <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().

|  |  |
| --- | --- |
| 1  2 | PreparedStatement <strong>createPreparedStatement</strong>  (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

|  |  |
| --- | --- |
| 1  2 | 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?

## <http://www.javarticles.com/2015/02/example-of-spring-callbacks-used-in-jdbctemplate.html>

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.

|  |  |
| --- | --- |
| 1 | void processRow(java.sql.ResultSet rs); |

|  |  |  |
| --- | --- | --- |
| Example of RowCallbackHandler ArticleRowCallbackHandler:  [?](http://www.javarticles.com/2015/02/example-of-spring-callbacks-used-in-jdbctemplate.html)   |  |  | | --- | --- | |  | package com.javarticles.jdbc.jdbctemplate;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  import org.springframework.jdbc.core.RowCallbackHandler;  public class ArticleRowCallbackHandler implements RowCallbackHandler {      private List<Article> aList;        public ArticleRowCallbackHandler() {          aList = new ArrayList<Article>();      }      public void processRow(ResultSet rs) throws SQLException {          aList.add(QueryUtils.extractArticleFromRs(rs));      }      public List<Article> getArticleList() {          return aList;      }} | |

# Spring AOP Interview Questions and Answers

Spring AOP is elegant feature of Spring Framework. It provides powerful to the cross cutting concern area into application. In this article I have collect Spring AOP (Aspect Oriented Programming) interview questions and answers.

**1. What is the concept of AOP? Which problem does it solve?**  
Aspect-Oriented Programming (AOP) is another way of thing to some areas of application i.e. cross cutting concern like security, logging and transaction. AOP is simple complement of OOP programming for different concerns. In OOP, the key unit of modularity is the class, whereas in AOP the unit of modularity is the aspect.

Aspect-Oriented Programming (AOP) enables modularization of cross-cutting concerns to solve following problems.

* To avoid tangling
* To eliminate scattering

Following Generic functionality that is needed in many places in your application

* Logging and Tracing
* Transaction Management
* Security
* Caching
* Error Handling
* Performance Monitoring
* Custom Business Rules

**AOP terminologies**

* Aspect
* Joint Point
* Advice
* Pointcut
* Introduction
* Target Object
* AOP Proxy
* Weaving

**2. What is a pointcut, a join point, an advice, an aspect, weaving, Introduction, Target Object, AOP Proxy?**  
**Pointcut**  
– An expression that selects one or more Join Points  
**Join Point**  
– A point in the execution of a program such as a method call or exception thrown  
**Advice**  
– Code to be executed at each selected Join Point  
**Aspect**  
– A module that encapsulates pointcuts and advice  
**Weaving**  
– Technique by which aspects are combined with main code  
**Introduction**  
-Spring AOP allows to introduce new interfaces (and a corresponding application) to any object advises.   
**Target Object**  
-An object is assisted by one or more respects. Also known as the object advised.   
**AOP Proxy**  
-AOP proxy is an object used to perform the contract area. This object is created by the AOP framework. In Spring AOP proxy is part of JDK dynamic proxy or proxy CGLIB.

**3. How does Spring solve (implement) a cross cutting concern?**

* Implement your mainline application logic
  + – Focusing on the core problem
* Write aspects to implement your cross-cutting concerns
  + – Spring provides many aspects out-of-the-box
* Weave the aspects into your application
  + – Adding the cross-cutting behaviors to the right places

**4. Which are the limitations of the two proxy-types?**  
Spring will create either **JDK**or **CGLib**proxies

1. **JDK Proxy**
   1. Also called dynamic proxies
   2. API is built into the JDK
   3. Requirements: Java interface(s)
   4. All interfaces proxied
2. **CGLib Proxy**
   1. NOT built into JDK
   2. Included in Spring jars
   3. Used when interface not available
   4. Cannot be applied to final classes or methods

**Popular Tutorials**

* [***Spring Tutorial***](https://www.dineshonjava.com/2012/06/spring-30-baby-step-to-learn.html)
* [***Spring MVC Web Tutorial***](https://www.dineshonjava.com/2012/12/spring-web-mvc-framework-chapter-38.html)
* [***Spring Boot Tutorial***](https://www.dineshonjava.com/2016/06/introduction-to-spring-boot-a-spring-boot-complete-guide.html)
* [***Spring Security Tutorial***](https://www.dineshonjava.com/2013/02/spring-security-take-baby-step-to-secure.html)
* [***Spring AOP Tutorial***](https://www.dineshonjava.com/2012/07/introduction-to-aop-in-spring.html)
* [***Spring JDBC Tutorial***](https://www.dineshonjava.com/2012/12/using-spring-jdbc-framework-chapter-32.html)
* [***Spring HATEOAS***](https://www.dineshonjava.com/2017/01/spring-hateoas-hypermedia-driven-restful-web-service.html)
* [***Microservices with Spring Boot***](https://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html)
* [***REST Webservice***](https://www.dineshonjava.com/2013/06/jax-rs-web-service-tutorial.html)
* [***Core Java***](https://www.dineshonjava.com/2013/01/core-java-baby-step-to-be-best-java-ian.html)
* [***Hibernate Tutorial***](https://www.dineshonjava.com/2012/03/hibernate-3-on-baby-steps.html)
* [***Spring Batch***](https://www.dineshonjava.com/2012/12/spring-batch-process-with-example.html)

**5. How many advice types does Spring support. What are they used for?**

* **Before advice:** Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
* **After returning advice:**Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
* **After throwing advice:** Advice to be executed if a method exits by throwing an exception.
* **After advice:**Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
* **Around advice:**Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.

**6. What do you have to do to enable the detection of the @Aspect annotation?**  
To use @AspectJ aspects in a Spring configuration you need to enable Spring support for configuring Spring AOP based on @AspectJ aspects, and autoproxying beans based on whether or not they are advised by those aspects.  
 **Enabling @AspectJ Support with Java configuration**  
To enable @AspectJ support with Java @Configuration add the @EnableAspectJAutoProxy annotation:

@Configuration

@EnableAspectJAutoProxy

public class AppConfig {

}

**Enabling @AspectJ Support with XML configuration**

To enable @AspectJ support with XML based configuration use the <aop:aspectj-autoproxy/> element:

<aop:aspectj-autoproxy/>

**7. Name three typical cross cutting concerns?**

1. Logging
2. Security
3. Transaction

**8. What two problems arise if you don’t solve a cross cutting concern via AOP?**  
Implementing Cross Cutting Concerns Without Modularization  
**• Failing to modularize cross-cutting concerns leads to two things**  
– Code tangling  
**• A coupling of concerns**  
– Code scattering  
**• The same concern spread across modules**  
 **Problem #1: Tangling**

public class RewardNetworkImpl implements RewardNetwork {

public RewardConfirmation rewardAccountFor(Dining dining) {

**//Non productive code or non functional code for business requirement**

**if (!hasPermission(SecurityContext.getPrincipal()) {**

**throw new AccessDeniedException();**

**}**

Account a = accountRepository.findByCreditCard(…

Restaurant r = restaurantRepository.findByMerchantNumber(…

MonetaryAmount amt = r.calculateBenefitFor(account, dining);

…

}

}

**Problem #2: Scattering**

public class JpaAccountManager implements AccountManager {

public Account getAccountForEditing(Long id) {

**//Non productive code or non functional code for business requirement**

**if (!hasPermission(SecurityContext.getPrincipal()) {**

**throw new AccessDeniedException();**

**}**

…

public class JpaMerchantReportingService

implements MerchantReportingService {

public List<DiningSummary> findDinings(String merchantNumber,

DateInterval interval) {

**//Non productive code or non functional code for business requirement -Duplicate across the application**

**if (!hasPermission(SecurityContext.getPrincipal()) {**

**throw new AccessDeniedException();**

**}**

…

**9. What does @EnableAspectJAutoProxy do?**  
To enable @AspectJ support with Java @Configuration add the @EnableAspectJAutoProxy annotation:

@Configuration

@EnableAspectJAutoProxy

public class AppConfig {

}

**10. What is a named pointcut?**  
A named pointcut can be declared inside an <aop:config> element, enabling the pointcut definition to be shared across several aspects and advisors.

<aop:config>

<aop:pointcut id="businessService" expression="execution(\* com.xyz.myapp.service.\*.\*(..))"/>

</aop:config>

**11. How do you externalize pointcuts? What is the advantage of doing this?**  
Externalize the pointcut to a named pointcut. Avoid to writing complex pointcut expression across the application.

**12. What is the JoinPoint argument used for?**  
Context provided by the JoinPoint parameter and Context about the intercepted point.

**13. What is a ProceedingJoinPoint?**  
An around advice is a special advice that can control when and if a method (or other join point) is executed. This is true for around advices only, so they require an argument of type ProceedingJoinPoint, whereas other advices just use a plain JoinPoint. ProceedingJoinPoint is used as an argument of the methods which hints for before, after, after throwing and around. ProceedingJoinPoint has the methods like getKind, getTarget, proceed etc.

**14. What are the five advice types called?**

* Before
* After
* AfterThrowing
* AfterReturning
* Around

**15. Which advice do you have to use if you would like to try and catch exceptions?**  
AfterThrowing

**16. Limitations of Spring AOP?**

* Can only advise non-private methods
* Can only apply aspects to Spring Beans
* Limitations of weaving with proxies
  + When using proxies, suppose method a() calls method b() on the same class/interface
* advice will never be executed for method b()

**17. What are the supported AspectJ pointcut designators in Spring AOP?**

* Execution
* This
* Target
* Args
* @target
* @args
* @within
* @annotation

**18. How to declare aspect in Spring AOP?**  
**In XML.**

<bean class="com.doj.aop.LoggingAspect" id="loggingAspect">

<!-- configure properties of aspect here -->

</bean>

**In Java**

@Aspect

@Component

class LoggingAspect{

//advice

//pointcut

}

**19. How to declare a pointcut in Spring AOP?**  
Find the below code snippet.

@Pointcut("execution(\* save(..))")

private void dataSave {}

**20. What do you understand by Load-time weaving (LTW) in Spring?**  
Load-time weaving (LTW) or Run time weaving is a process of weaving AspectJ aspects into the classes of the application when the classes are being loaded in JVM.

# An Interview Question on Spring Singletons

Please tell me what the output of the following program would be."

Spring.xml:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="scopeTest" class="com.example.scope.Scope" scope="singleton">

<property name="name" value="Shamik Mitra"/>

</bean>

<bean id="scopeTestDuplicate" class="com.example.scope.Scope" scope="singleton">

<property name="name" value="Samir Mitra"/>

</bean>

</beans>

Scope.java:

package com.example.scope;

public class Scope {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Scope [name=" + name + "]";

}

}

Main class:

package com.example.scope;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

ApplicationContext ctx = new ClassPathXmlApplicationContext(

"configFiles/Scope.xml");

Scope scope = (Scope) ctx.getBean("scopeTest");

Scope scopeDuplicate = (Scope) ctx.getBean("scopeTestDuplicate");

System.out.println(scope == scopeDuplicate); //false

System.out.println(scope + "::" + scopeDuplicate);

}

}

Output:

Reference Check ::false

Scope [name=Shamik Mitra]::Scope [name=Samir Mitra]

# A Spring Singleton does not work like a Java Singleton

The reference check will return false, which means Spring Singletons don't work like they said earlier. (**A few**)

**===LIST OF PROPOGATIONS IN AOP====**

**MANDATORY**  
Support a current transaction, throw an exception if none exists.

Does not start a new Transaction, just checks whether a transaction is active (must be inside either another @Transactional method call or a programmatically created transaction)

**NESTED**  
Execute within a nested transaction if a current transaction exists, behave likePROPAGATION\_REQUIRED else.

Start a nested transaction if a transaction exists, start a new transaction otherwise.

**NEVER**  
Execute non-transactionally, throw an exception if a transaction exists.

Does not start a transaction. Fails if a transaction is present.

**NOT\_SUPPORTED**  
Execute non-transactionally, suspend the current transaction if one exists.

Does not start a transaction. Suspends any existing transaction.

**REQUIRED**  
Support a current transaction, create a new one if none exists.

If a transaction exists, use that, if not, create a new one. **In 95% of cases, this is what you need.**

**REQUIRES\_NEW**  
Create a new transaction, suspend the current transaction if one exists.

Always creates a new transaction, no matter if an existing transaction is present. If there is, it will be suspended for the duration of this method execution.

**SUPPORTS**  
Support a current transaction, execute non-transactionally if none exists.