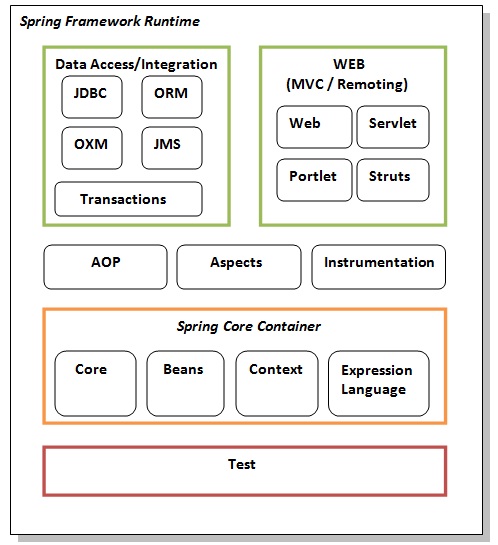
**Spring IOC**

**Q-1: What are the different modules in Spring framework?**

**A:** The Spring framework comprises of many modules such as **1-**core, **2-**beans,

**3-**context, **4-**expression language, **5-**AOP, **6-**Aspects, **7-**Instrumentation, **8-**JDBC, **9-**ORM, **10-**OXM,**11-** JMS, **12-**Transaction, **13-**Web, **14-**Servlet, **15-**Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



**1-Test**

This layer provides support of testing with JUnit and TestNG.

**2-Spring Core Container**

The Spring Core container contains core, beans, context and expression language (EL) modules.

**3-Core and Beans**

These modules provide IOC and Dependency Injection features.

**4-Context**

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

**5-Expression Language**

It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, retrieval of objects by name etc.

**6-AOP, Aspects and Instrumentation**

These modules support aspect oriented programming implementation where you can use Advices, Pointcuts etc. to decouple the code.

**The aspects** module provides support to integration with AspectJ.

**The instrumentation** module provides support to class instrumentation and classloader implementations.

**7-Data Access / Integration**

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

**8-Web**

This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.

**Q-2: What is Spring and What are benefits of using spring?**

**A:** Spring is a lightweight framework.it can be thought of as a framework of framwork because it provide support to various framework such as Struts,Hibernate,EJB,JSF etc. Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

There are many advantages of Spring Framework. They are as follows:

**1->Predefined Template**

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies.

So there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of JdbcTemplate, you don't need to write the code for exception handling, creating connection, creating statement, committing transaction, closing connection etc. You need to write the code of executing query only. Thus, it save a lot of JDBC code.

**2->Loose Coupling**

The Spring applications are loosely coupled because of dependency injection.

**3->Easy to test**

The Dependency Injection makes easier to test the application.

The EJB or Struts application require server to run the application but Spring framework doesn't require server.

**4->Lightweight**

Spring framework is lightweight because of its POJO implementation.

The Spring Framework doesn't force the programmer to inherit any class or implement any interface.

**5->Fast Development**

The Dependency Injection feature of Spring Framework and

it support to various frameworks makes the easy development of JavaEE application.

**6->Powerful abstraction**

It provides powerful abstraction to JavaEE specifications such as JMS, JDBC, JPA and JTA.

**7->Declarative support**

It provides declarative support for caching, validation, transactions and formatting.

**8-Inversion of control (IOC):** Loose coupling is achieved in Spring, with the [Inversion of Control technique](http://www.javacodegeeks.com/2011/08/what-is-dependency-inversion-is-it-ioc.html). The objects give their dependencies instead of creating or looking for dependent objects.

**9-Aspect oriented (AOP):** [Spring supports Aspect oriented programming](http://www.javacodegeeks.com/2011/01/aspect-oriented-programming-spring-aop.html) and separates the middle level services from High level services.

Transaction , security, logging, messaging are the middle level services and core

Business logic is high level services and IO, threading, Networking are the low level services and low level services are freely provided by server.

**10-MVC Framework:** Spring’s web framework is a well-designed [web MVC framework](http://www.javacodegeeks.com/2011/02/spring-mvc-development-tutorial.html), which provides a great alternative to web frameworks.

**11-TransactionManagement:** 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).

**Q-3: What is Spring configuration file?**

**A:** 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.

<?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="add1" class="Address">

<constructor-arg value= "lucknow"/>

<constructor-arg value= "UP"/>

<property name="country" value="India"/>

</bean>

<bean id="add2" class="Address">

<property name="city" value= "Bhopal"/>

<property name="state" value= "MP"/>

<property name="country" value= "India"/>

</bean>

</beans>

**Q-4: What is IOC(Inversion of Control)**

**A:** - IOC stands for Inversion of Control pattern.   
- It is also called as dependency injection.   
- This concept says that you do not create your objects but describe how they should be created.   
- Similarly, you do not directly connect your components and services together in code but describe which services are needed by which components in a configuration file.

-.

**Q-5: What is the diff b/w IOC and Dependency Injection.**

**A:** Inversion of Control (IoC) is a general concept and desigen pattern, and it can be expressed in many different ways and Dependency Injection is merely one concrete example of Inversion of Control.

**Q-6: What is Dependency Injection?**

**A:** the process of inetializing of Bean Dependencies is called Dependency injection.

**Q-7: What are the different types of (IOC)dependency injection?**

**A:** Three ways to perform Dependency Injection in Spring framework

Spring framework provides two ways to inject dependency

1-By Constructor

2-By Setter method

3- Field Injection (by Annotation)

**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.

**Syntax:**

<bean id="add1" class="Address">

<constructor-arg value= "lucknow"/>

<constructor-arg value= "UP"/>

<constructor-arg value= "India"/>

</bean>

**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 **Syntax:**

<bean id="add3" class="Address">

<property name="city" value= "Bhopal"/>

<property name="state" value= "MP"/>

<property name="country" value= "India"/>

</bean>

**Q-8: Which DI would you suggest Constructor-based or setter-based DI?**

**A:** Since you can mix both, Constructor- and Setter-based DI, it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies

**Q-9:** **Difference between constructor and setter injection**

**A:** There are many key differences between constructor injection and setter injection.

**1-Partial dependency:->** can be injected using setter injection but it is not possible by constructor. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.

**2-Overriding:->** Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.

**3-Changes:->** We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

**Q-10: When to use Dependency Injections?**

**A:** There are different scenarios where you Dependency Injections are useful.

* You need to inject configuration data into one or more component.
* You need to inject the same dependency into multiple components.
* You need to inject different implementation of the same dependency.
* You need to inject the same implementation in different configuration.
* You need some of the services provided by container.

**Q-11: When you should not use Dependency Injection?**

**A:** There were scenarios where you don’t need dependency injections e.g.

* You will never need a different implementation.
* You will never need different configurations.

**Q-12: What are the benefits of IOC?**

**A:** The main benefits of IOC or dependency injection are:

1-It minimizes the amount of code in your application.

2-It makes your application easy to test as it doesn't require  any singletons or JNDI lookup mechanisms in your unit test cases.

3-Loose coupling is promoted with minimal effort and least intrusive mechanism.

4-IOC containers support eager instantiation and lazy loading of services

**Q-13: What is Spring IOC container?**

**A:** The IOC container is responsible to instantiate, configure and assemble the objects. The IOC container gets informations from the XML file and works accordingly.

The main tasks performed by IOC container are:

1- to instantiate the application class

2- to configure the object

3- to assemble the dependencies between the objects

**Q-14: What are types of IOC containers? Explain them.**

**A:** There are two types of IoC containers:

* **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 container 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.

**Q-15: Give an example of BeanFactory implementation.**

**A:** The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

**Resource resource=new ClassPathResource("applicationContext.xml");**

**BeanFactory factory=new XmlBeanFactory(resource);**

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

**Q-16: What are the common implementations of the ApplicationContext?**

**A:** The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

**ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");**

The constructor of ClassPathXmlApplicationContext class receives string,

so we can pass the name of the xml file to create the instance of ApplicationContext.

**Q-17: What is the difference between Bean Factory and ApplicationContext?**

**A:**

|  |  |
| --- | --- |
| **BeanFactory** | **ApplicationContext** |
| BeanFactory is an interface which as one concrete sub class called  XMLBeanFactory | ApplicationContext is an interface which as  3 concrete sub class called   1. ClassPathXmlApplicationContext 2. FilesSystemXmlApplicationContext 3. XmlWebApplicationContext |
| Bean configured will be loaded lazily  by default | Bean configured will be loaded aggressively  By default |
| It does not support Annotation | It support Annotation |
| It does not support BeanPostProcessor | It does support BeanPostProcessor |
| It does not support event publishing | It does support event publishing |
| It not resolves message bundles | It resolves message bundles |

**Q-18: What are Spring beans?**

**A:** Every classyou are writing in spring application can be called as Spring Bean.

**Q-19: What does a Spring bean definition contain?**

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

**1-**How to create a bean

**2-**Bean's lifecycle details

**3-**Bean's dependencies

**Q-20: How do you provide configuration metadata to the Spring Container?How do add a bean in spring application**

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

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

**Q-21: How do add a bean in spring application?**

**A:**

<bean id="helloWorld" class="com.tutorialspoint.HelloWorld">

<property name="message" value="Hello World!"/>

</bean>

**Q-22: What bean scopes does Spring support? Explain them.**

**A**: Bean Instance created by Spring Container can be in one of the following

Scopes.

1- singleton

2- prototype

3- request

4- session

5- global-session

Usage:

<bean id= " " class=" " scope=" "/>

**1-singleton->**

1- when bean scope is singeton then onlyone instance will be created for that bean and the same instance will be returned when you call getBean() method.

2- singleton is the default scope in the ApplicationContext container.

3- When scope is single-ton then default loading type is aggressive loading.

**2-prototype->**

1-when bean scope is prototype then every then every time a new instance will be created for that bean when you call getBean() method.

2-When scope is prototype then default loading type is lazy loading.

**3-request->**

1- request scope is equals to HttpServletRequest in the web application

**4-session->**

1- session scope is equals to HttpSession in the web application

**5-global-session->**

1-global-session scope is equals to session in the portlet based web application

**Q-23: What is default scope of bean in Spring framework?**

**A:** The default scope of bean is Singleton for Spring framework.

**Q-24: What is default loading in Spring framework?**

**A:** aggressive loading. Because the default scope is Singleton.

**Q-24: How you will decide when to use prototype scope and when singleton scope bean?**

**A:** You should use the prototype scope for all beans that are stateful and the singleton scope should be used for stateless beans.

**Q-25: Are Singleton beans thread safe in Spring Framework?**

**A:** No, singleton beans are not thread-safe in Spring framework.

**Q-26: What is the loading Strategies in Spring Framework.**

**A:** Bean configured in the Spring Context xml can be loaded in two ways.

1- Aggressive loading or Eager loading

2- lazy loading

Usage:

<bean id= " " class=" " scope=" " lazy-init=""/>

**1-Aggressive loading or Eager loading->**

in the case of aggressive loading,all the Beans will be loaded,instantiated and

inetialized by the container at the container start-up.

<bean id= " " class=" " scope=" " lazy-init="false"/>

**2-lazy loading->**

in the case of lazy loading,all the Beans will be loaded,instantiated and

inetialized when you or container try to use them by calling getBean() method.

<bean id= " " class=" " scope=" " lazy-init="true"/>

**Q-27: Explain Bean lifecycle in Spring framework.**

**A:** Following is sequence of a bean lifecycle in Spring:

* **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 postprocess of bean. If there are any bean BeanPostProcessors[ASSOCIATED[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.tutorialspoint.com/#4983693)](http://www.tutorialspoint.com/#4983693) 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[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.tutorialspoint.com/#77297327)](http://www.tutorialspoint.com/#77297327) 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 .

**Q-28: What are inner beans in Spring?**

**A:**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.

**Q-29: How can you inject a Java Collection in Spring?**

**A:**Spring offers the following types of [collection configuration elements](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-collections-list-set-map-and-properties-example/):

* The <list> type is used for injecting a list of values, in the case that duplicates are allowed.
* The <set> type is used for wiring a set of values but without any duplicates.
* The <map> type is used to inject a collection of name-value pairs where name and value can be of any type.
* The <props> type can be used to inject a collection of name-value pairs where the name and value are both Strings.

**Q-30: What is wiring.**

**A:** 1-Wiring is the process of injecting the Dependencies of the Bean.

2-Wiring can be done in two ways

1- Explicit wiring

2- Implicit wiring or Auto wiring

**1- Expliciet wiring->**

in the case of explicit wiring,you have to specify the bean Dependencies explicitly then container will inject those Dependencies.

<beans..>

<bean id = "ao" class= "A"/>

<bean id = "bo" class= "B"/>

<bean id="hello" class="Hello">

<property name="aobj" ref="ao"/>

<property name="bobj" ref="bo"/>

</bean>

</beans>

**2- Auto wiring->**

in the case of Auto wiring , Spring Container can detect the Bean Dependencies

automatcally and injects those Dependencies.

<beans..>

<bean id = "ao" class= "A"/>

<bean id = "bo" class= "B"/>

<bean id="hello" class="Hello" autowire="yyy"/>

</beans>

note->

1-Autowiring can't be used to inject primitive and string values.

It works with reference only.

2-It requires the less code because we don't need to write the code to inject

the dependency explicitly.

**Q-31: What is bean auto wiring?**

**A:** in the case of Auto wiring , Spring Container can detect the Bean Dependencies automatcally and injects those Dependencies.

**Q-32: What are different Modes of auto wiring?**

**A:** **There are many autowiring modes:**

**1 ->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.

**2 ->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.

**3 ->constructor->**The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.

**1->byName:**

1-when autowiring mode is "byName" then Spring Container checks whether any bean instance running in the Container whose name(or id) is same as bean property(variable) name or not.

2-when bean is found with the matching name then it will be injected otherwise bean property remains uninjected.

3-Bean will be instantiated using Default constructor.

4-Dependent Bean instances will be detected using bean name.

5-Detcted bean instances will be injected through setter injection only.

**2-byType->**

1- when autowiring mode is "byType" then Spring Container checks whether any bean instance running in the Container whose Type is same as bean property Data type or not.

2- in this you may get three cases:

a-> when exactly one bean is found with the matching Data type then it will

be injected.

b-> when no beans is found with the matching then bean property will not be

injected.

c-> when two or more beans is found with the matching Data type then exception will be thrown called UnsatisfiedDependencyException.

3-Bean will be instantiated using Default constructor.

4-Dependent Bean instances will be detected using bean name.

5-Detcted bean instances will be injected through setter injection only.

**Q-33: What are the limitations with autowiring?**

**A:** Limitations of autowiring are:

* **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.

**Q-34: Can you inject null and empty string values in Spring?**

**A:** Yes.

**Q-35: What is Annotation-based autowiring?**

**A:**when you want to use annotation based autowiring,you need to do the following tasks.

1-You must enable context namespace

Add <context:annotation-config/>

2-when you use **@Autowired ,** then by default,beans will be detected based on “byType” process and inject them.

3-By default,its functionality is same as autowire=”byType”.

4-when you want to detect the beans based on byName process then you need to **@Qualifier** along with @Autowired.

5-when you use **autowire=”byType”** the beans will be detected based on Data type and injects using setter methods whereas when you use **@Autowired** then beans will be detected based on Data type and injects without using setter methods .you can use setter method also by specifying the property variable in XML file

6-when you use **autowire=”byName”** the beans will be detected based on bean name and injects using setter methods whereas when you use “ **@Autowired &@Qualifier** then beans will be detected based on bean name and injects without using setter methods. you can use setter method also by specifying the property variable in XML file

**Q-36: What is the use of required attribute.**

**A:**@Autowired(required=false/true)->default is **true** and it means if matching

bean found then it is OK otherwise exception will be thrown

if value is false and it means if matching bean found then it is OK otherwise

it will not inject and gives result null.

**Q-37: How do you turn on annotation wiring?**

**A:** Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file by configuring <context:annotation-config/>.

**Q-38: What does @Autowired annotation mean?**

**A: @Autowired** means beans will be detected based on Data type and injects without using setter methods .you can use setter method also by specifying the property variable in XML file

**Q-39: What does @Qualifier annotation mean?**

**A:** you use “ **@Autowired &@Qualifier** then beans will be detected based on bean name and injects without using setter methods. you can use setter method also by specifying the property variable in XML file

**Q-40: What does @Required annotation mean?**

**A:** The **@Required** annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time otherwise the container throws a **BeanInitializationException** exception. Below is an example to show the use of @Required annotation.

**Q-41: What are the JSR-250 Annotations? Explain them.**

**A:**JSR(java specification required) is a specification document.

following annotations provided in javax.annotation package

1- @PostConstruct init()

2- @PreDestroy destroy()

3- @Resource

Note: when you want to use JSR-250 Annotations you must add javee.jar file to project build path

**1- @PostConstruct🡪**

1-you can mark any method with @PostConstruct Annotation to call that method after creating the instance.

2-Method which is marked with @PostConstruct Annotation contains the code for inetializing bean instance with the required resources.

**2- @PreDestroy🡪**

1-you can mark any method with @PreDestroy Annotation to call that method before destroying the instance.

2-Method which is marked with @PreDestroy Annotation contains the code for

cleaning the resources inetialized with bean instance.

**3- @Resource🡪**

1-when you use @Resource then bean will be detected either based on byName or byType process and injects them.

🡪 when name attribute is specified for @Resource then use "byName" process.

🡪when name attribute is not specified for @Resource then use "byType" process

**Q-42: What is Spring Java Based Configuration? Give some annotation example.**

**A:** Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.

**For example:** Annotation **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions.

The **@Bean** annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

**Q-43: How is event handling done in Spring?**

**A:**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.

**Q-44: Describe some of the standard Spring events.**

**A:** Spring provides the following standard 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[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.tutorialspoint.com/#91598672)](http://www.tutorialspoint.com/#91598672) 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.

**Q-41: Expalin spring callbacks or lifecycle methods.**

**Spring Data Access**

**Q-42: What is JdbcTemplate in Spring? And how to use it?**

**A:** jdbcTemplate class provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

**Some method of JdbcTemplate**

1. int **update**(sql) ->delete
2. int **update**(sql,args) ->insert,update,delete
3. int **update**(sql,args,argTypes) ->insert,update,delete

1. Object **queryForObject**(sql,args,rowMapper) ->Select
2. Object **queryForObject(**sql,args,argTypes,rowMapper) ->Select
3. Object **queryForObject**(sql,args,Class) ->Select
4. public T **execute(String sql, PreparedStatementCallback action) ->**

executes the query by using PreparedStatement callback.

1. public T **query(String sql, ResultSetExtractor rse) ->**

is used to fetch records using ResultSetExtractor.

1. public List **query(String sql, RowMapper rse)** ->

is used to fetch records using RowMapper.

### \*ResultSetExtractor Interface->

### ResultSetExtractor interface can be used to fetch records from the database. It accepts a ResultSet and returns the list.

**Q-44: What is RowMapper**

**A:** we can use RowMapper interface to fetch the records from the database using **query()** method of **JdbcTemplate**class

RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the collection.

**How to write RowMapper->**

1. Write your own class by implementing RowMapper interface
2. Override the following method.

public Object mapRow(ResultSet rs, int rowNum) throws SQLException();

1. Implement the code inside the mapRow() to move the data from ResultSet to Customer class

import java.sql.ResultSet;

import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class CustomerRowMapper implements RowMapper

{

**public Object mapRow(ResultSet rs, int rowNum) throws SQLException** {

Customer customer = new Customer();

customer.setCustId(rs.getInt("CUST\_ID"));

customer.setName(rs.getString("NAME"));

customer.setAge(rs.getInt("AGE"));

return customer;

}

}

**Method in jdbc template class ->**

public Customer findByCustomerId(int custId){

String sql = "SELECT \* FROM CUSTOMER WHERE CUST\_ID = ?";

Customer customer = (Customer)getJdbcTemplate().queryForObject(

sql, new Object[] { custId }, new CustomerRowMapper());

return customer;

}

}

**Q-43: What are the advantages of JdbcTemplate in spring?**

**A: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.

**Q-44: What is Hibernate Template and Explain the method.**

**A: 1-** Hibernate Template is provided which centralizes the Hibernate client code.

**Usage:**

hibernateTemp.save(cust);

**2-** In Spring Data Access ,There is one root exception called DataAccessException which unchecked or runtime .Because of Unchecked exceptions , you need to write try/catch blocks for every program.

**3-**There is clear categorization of exception in Spring Data Access.

**Important Method of Hibernate Template->**

1. Serializable save(Object )
2. Void update(Object )
3. Void update(Object , LockMode )
4. Void delete(Object )
5. Void delete(Object , LockMode )
6. Object load(Class , Serializable )
7. Object load(Class , Serializable ,LockMode )
8. List find( hql )
9. List find( hql ,Object)

**Q-44: What are classes for spring JDBC API? Anf Type of jdbc Template**

A:

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

**Q-45: Explain about the Spring DAO support**

**A:**The [Data Access Object (DAO) support in Spring](http://www.javacodegeeks.com/2012/09/spring-dao-and-service-layer.html) is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows us to switch between the persistence technologies fairly easily and to code without worrying about catching exceptions that are specific to each technology.

**Q-46: What are the ways to access Hibernate by using Spring?**

**A:** 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.

**Q-47: How would you integrate Spring with Hibernate using HibernateDaoSupport?**

**A:** These are the Steps to integrate the Spring with Hibernate

**1- Configure DriverManagerDataSource with the following four properties in the Spring Configuration file:**

<bean id = “dataSource”

class = “org.springframework.jdbc.datasource.DriverManager.DataSource”>

<property name="driverClassName” value = “com.mysql.jdbc.Driver/>

<property name="url" value = jdbc:mysql://localhost/subhag/>

<property name="username" value = root/>

<property name="password" value=root/>

</bean>

**2- Configure LocalSessionFactoryBean class by injecting following properties:**

<bean id="mysessionFactory"

**class**="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

        <property name="dataSource" ref="dataSource"></property>

        <property name="mappingResources">

         <list>

         <value>com/jlcindia/spring/hibernate/customer.hbm.xml</value>

         </list>

        </property>

        <property name="hibernateProperties">

            <map>

                <entry key="hibernate.dialect">org.hibernate.dialect.MySQLDialect/>

                <entry key="hibernate.hbm2ddl.auto" value=”update”/>

                <entry key="hibernate.show\_sql" value = **true**/>

              </maps>

        </property>

    </bean>

**3-Configure HibernateTemplate class by injecting SessionFactory through constructor injection**

 <bean id="template"

**class**="org.springframework.orm.hibernate3.HibernateTemplate"

**autowire = “constructor”**>

   </bean>

**4-Create the package called com.jlcindia.spring.hibernate**

**5-inside package create the Persistence class called Customer.java**

**6- Create the Hibernate Mapping Document**

**7- Write the CustomerDAO interface with the required database operations**

**8- Write the sub class for CustomerDAO interface called HibernateCustomerDAO and do the following**

1. **Inject HibernateTemplate into HibernateCustomerDAO**
2. **Override the CustomerDAO methods in HibernateCustomerDAO using the appropriate methods of HibernateTemplate.**

**9-Write the client and run it**

**Q-48: What are ORM's Spring supports ?**

**A:** Spring supports the following ORM's :

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

**Q-49: What are the types of the transaction management Spring supports?**

**A:** Spring supports two types of transactionManagement

* **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 transactionManagement from the business code. You only use annotations or XML based configuration to manage the transactions

**Q-50 What are the benefits of the Spring Framework’s transaction management?**

**A:1-**It provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.

**2-**It provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.

**3-**It supports declarative transaction management.

**4-**It integrates very well with Spring’s various data access abstractions.

**Q-51: Which of the above transaction management type is preferable?**

Most users of the Spring Framework choose **declarative transaction** management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container. 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.

**Q-52: When do you use programmatic and declarative transaction management A:** Programmatic transaction management is usually a good idea only if you have a small number of transactional operations.   
On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure.

**Spring MVC**

**Q-53: What is Spring MVC framework?**

**A:** 1-Spring MVC is Web Framework like Struts1,Struts2 and JSF

2-Spring MVC is used to develop web applications easily and quickly with less maintenance

3-Spring MVC covers Web Layer which includes Presentation layer and Controller layer

4-Spring MVC is implemented based on:

* 1. MVC Architecture
  2. Front Controller Desigen Pattern
  3. Servlet and JSP

**Q-54: What is the flow of Spring MVC framework**

**A: (see diagram from jlc-book)**

These aresome steps of Spring MVC framework

1. DispatcherServlet takes the incoming request
2. DispatcherServlet contacts the Handler mapping with the incoming request URI(/course.jlc)
3. Handler Mapping identifies and returns the Controller**(Courese Controller)** specified for the request URI(/course.jlc)
4. DispatcherServlet Identifies and invoke one or more Handler Interceptors registered with Spring Container if any and then DispatcherServlet invokes the Controller

**getCourse() method of CourseController with Annotation**

1. After finishing Controller method execution,all the registered Handler Interceptor will be called in the reverse one by one

**Finally DispatcherServlet gets the view logical name with Annotation**

1. DispatcherServlet contacts the ViewResolver with the view logical name(show)
2. DispatcherServlet gets the view(/show.jsp) from ViewResolver
3. DispatcherServlet forwards the identified view(/show.jsp) to the client

**Q-55: What is front controller class in Spring MVC?**

**A: Front-Controller** is a initial level of contract point for handling a request. The front controller provides a centralized entry point for that controls and manages web request handling. the front controller also reduce your java code and business logic by promoting code reuse ability across the requests.

A front controller is defined as “a controller which handles all requests for a Web Application.” **DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.**

When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring’s web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

**Q-A: How DispatcherServlet identified which controller has to invoke**

**Q-B: how viewResolver decides which jsp to invoke**

**Q-56:Explain the Main Elements of Spring MVC**

**A:** There are some various component of Spring MVC

**Presentation Layer Components**

1. JSP/EL/JSTL
2. Spring Form Tag Library
3. Message Bundles
4. Command
5. View Resolvers

**Controller Layer Components**

1. DispatcherServlet
2. HandlerMappings
3. HandlerInterceptors
4. Controller

**Q-57: What is a DispatcherServlet .**

**A:** DispatcherServlet is the front Controller Component which is responsible for the following :

* 1. Receives the incoming request
  2. Process the request completely
  3. Deliver the response

You should configure DispatcherServlet in web.xml as follows:

<servlet>

<servlet-name>jlcindia</servlet-name>

<servlet-class> **org.springframework.web.servlet.DispatcherServlet** </servlet-class>

<load-on-startup>0</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>jlcindia</servlet-name>

<url-pattern>\*.jlc</url-pattern> (Any servlet that and with .jlc )

</servlet-mapping>

**Q-58: What is Handler Mapping. Explain the type of Handler Mapping.**

**A:1-**Handler Mapping are responsible for identifying the corresponding controller for incoming request URI

**2-**All the Handler Mapping provided are subclasses of HandlerMapping interface.

**3-**Following are the various Handler Mapping provided in Spring MVC

1. BeanNameUrlHandlerMapping
2. DefaultAnnotationUrlHandlerMapping
3. SimpleUrlHandlerMapping
4. ControllerBeanNameHandlerMapping
5. ControllerClassNameHandlerMapping

**1-BeanNameUrlHandlerMapping->**

**1-**This is default Handler Mappings which will be registered by the Spring Container Automatically

**2-**This Handler Mapping checks whether any bean available whose name is same as incoming request

Index.jsp jlcindia-servlet.xml

<bean name = **“/hello.jlc”** class = “…HelloController”>

<bean name = **“/hai.jlc”** class = “…HaiController”>

<bean name = **“/Hay.jlc”** class = “…HelloController”>

<a href = **“hello.jlc”**>Hello guys</a>

<a href = **“hai.jlc”**>Hai guys</a>

<a href = **“Hay.jlc”**>Hello guys</a>

**2-BeanNameUrlHandlerMapping->**

**1-**This is default Handler Mappings which will be registered by the Spring Container Automatically

**2-**This Handler Mapping checks whether any method available in the controller class whose @RequestMapping value is same as incoming request

Index.jsp HelloController.java

@Controller

**Class HelloController**

**{**

@RequestMapping(values = **“/hello.jlc”**)

Public String showHello(…) {}

@RequestMapping(values = **“/hai.jlc”**)

Public String showHai(…) {}

@RequestMapping(values = **“/hay.jlc”**)

Public String showHay(…) {}

<a href = **“hello.jlc”**>Hello guys</a>

<a href = **“hai.jlc”**>Hai guys</a>

<a href = **“hay.jlc”**>Hello guys</a>

**Q-58: What is Handler Interceptors.**

**A:** Spring MVC Interceptors are like Servlet Filter and allow use to intercept client request and process it. We can intercept client request at three places🡪

preHandle , postHandle and afterCompletion

Handler Interceptors will be invoked before and after the controller invocation

for performing pre-processing and post-processing tasks.

You can have one or more Handler Interceptors in the application.

**Steps to write Handler Interceptors->**

We can create spring interceptors by implementing **HandlerInterceptor** interface or by extending abstract class **HandlerInterceptorAdapter**

**1-**Write your Interceptor class by implementing HandlerInterceptor interface.

**2-**Mark your Interceptor class with @Component annotation.

**3-**Override the following methods in your Interceptor class:

**a->** public boolean preHandle(request,response,Object handler)

**b->** public void postHandle(request,response,Object handler,MadelAndView)

**c->** public void afterCompletion(request,response,Object handler,Exception e)

**Ex:**

@Component

Public class MyInterceptor1 implements HandlerInterceptor

{

public boolean preHandle(HttpServletRequest req , HttpServletResponse res,

Object obj)

{

return true;

}

public void postHandle(HttpServletRequest req , HttpServletResponse res,

Object obj ,ModelAndView mav)

{

}

public void afterCompletion(HttpServletRequest req , HttpServletResponse res,

Object obj ,Exception e)

{

}

4->you can configure your Interceptor classes in Spring Configuration Document in two ways:

**a->** **Two apply the Interceptor for all the incoming request URIs**

<mvc:interceptors>

<bean class = “com.jlcindia.spring.mvc.MyInterceptor1”/>

</mvc:interceptors>

**b->** **Two apply the Interceptor for specific incoming request URIs**

<mvc:interceptors>

<mvc:mapping path=”/hai.jlc”/>

<bean class = “com.jlcindia.spring.mvc.MyInterceptor1”/>

</mvc:interceptors>

**Method of HandlerInterceptor Interface->**

**1-preHandle()->**

1-This method will be called just before the controller method.

2-This method returns Boolean value

3-When preHandler() method returns true then Interceptors or Controller in HandlerExecutionChain will be called.

4- When preHandler() method returns false then Control will be returned to Client without invoking Interceptors or Controller in HandlerExecutionChain and No view will be rendered.

**2-postHandle()->**

This method will be called immediately after controller method execution

**3-afterCompletion()->**

This method will be called just before sending the response.

**Q-59: What is View Resover class .Explain the type of View Resolver.**

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

Spring MVC provides the ViewResolver which resolve view logical name to actual views.

**Following are the list of View Resolver provided in Spring MVC**

1-InternalResourceViewResolver (only this is required)

2- XMLViewResolver

3- ResourceBundleViewResolver

4-ContentNegotiatingViewResolver (Required when we use web services)

5- BeanNameViewResolver

6- UrlBasedViewResolver

7- FreeMarkerViewResolver

8-JasperReportViewResolver

**Q-60: Which ViewResolver class is widely used?**

**A:**The **org.springframework.web.servlet.view.InternalResourceViewResolver**

class is widely used

Ex:

**<**bean class = ”org.springframework.web.servlet.view.InternalResourceViewResolver”**>**

**<**property name = “viewClass”

value = ”org.springframework.web.servlet.view.InternalResourceView”>

<property name = “prefix” value = “/” />

<property name = “suffix” value = “.jsp” />

</bean>

**Q-61:What is a MultipartResolver and when its used?**

**A:**MultipartResolver use to handle file upload in web application.

<bean id = “multipartResolver”

class =”org.springframework.web.multipart.commons.Commons MultipartResolver”>

<property name = “maxUploadSize” value = “200000”>

</bean>

**Q-62: What is Controllers in Spring MVC.**

**A:** Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view.

**1-**Controller is the last component in the Controller layer from where Business layer starts.

**2-**Controller is mainly responsible for the following

1. Collects the client submitted data from Command Object
2. Call the Model Component
3. If Error is occurred then handle that in two ways
4. Throws as Exception
5. Add as an Error
6. If Data is returned successfully without any error then store that in any scope and return View logical name

**3-**Only One instance will be created per Controller for all the users

**4-**When you are writing Controller class with Annotation Configuration then you need to do the following steps

1. Write your own Controller class without extending any Built-In Controllers.
2. Mark the Controller with **@Controller** annotation.
3. Implement the required methods in the Controller Class to process the Incoming Request URI

**@RequestMapping(value=”/login.jlc”)**

String showLoginPage(Map map){}

**@RequestMapping(value=”/verifyUser.jlc”)**

String verifyUser(User , Errors){}

1. No need to Configure the Controller in Spring Configuration Document
2. Make sure that <context:component-scan> tag is present in Spring Configuration Document.

**Q-63: What is WebApplicationContext ?**

**A:** The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications. It differs from a normal ApplicationContextin that it is capable of resolving themes, and that it knows which servlet it is[ASSOCIATED[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.tutorialspoint.com/#54703262)](http://www.tutorialspoint.com/#54703262)with

**Q-64: Explain in Spring Form Tag Library.**

**A:** Spring MVC provides a JSP tag library (Spring Form) for making it easier to bind form elements to Model data. Spring Framework also provides you with some tags for evaluating errors, setting themes and outputting internationalized messages.

It is use basically use for Presentation Purpose.

|  |  |  |  |
| --- | --- | --- | --- |
| <form:form/> | <form:input/> | <form:lable/> | <form:password/> |
| <form:hidden/> | <form:checkbox/> | <form:checkboxes/> | <form:errors/> |
| <form:option/> | <form:options/> | <form:radiobutton/> | <form:radiobuttons/> |
| <form:select/> | <form:textarea> |  |  |

**Q-65: What is Resource/Message Bundles in Spring Framework.**

**A:** Any web application with users all around the world, internationalization ( or localization is very important for better user interaction.

In Spring, you can use ResourceBundleMessageSource to resolve text messages from properties file, base on the selected locales. so if want to use chaines Language ,Hindi Language then by using message.properties file you can do.

it also useful for accessing "Labels and Error Message"

**1-**you can centralize Labels and Error Message in message bundle

**2-**you can write one or more message bundels in your web application.

**Syntax:**

basename\_CountryISOCode\_LanguageISOCode.properties

basename\_LanguageISOCode.properties

**3-**for accessing any message from message bundle,you need to register **ResourceBundleMessageSource** in Spring Configuration Document with basename

<bean id = ”messgaeSource”

class = “org.springframework.context.support. ResourceBundleMessageSource”>

<property name = “basename” value = “message”/>

</bean>

**A-> Accessing Labels**

Ex:

1-> Message.properties

Jlc.title = JLC(EN)

2-> Message.properties

Jlc.title = JLC(HI)

Use the required keys in JSP using JSTL tahs as follows:

<%@ taglib prefix=”fmt” uri = <http://java.sun.com/jsp/jstl/fmt>%>

<fmt:message key=”jlc.title”>

**B->Accessing Error Messages**

Ex:

1-> Message.properties

errors.username.required = Username is required.

errors.required = {0} is required.

Error.legnth = {0} length must be b/w {1} and {2}

**In validate() method**

**1-**errors.rejectValue(“username”,”errors.username.required”,null,”Username is Mandatory”)

Display Username is required

**2-**errors.rejectValue(“username”,”uname.errors.required”,null,”Username is Mandatory”)

Display Username is Mandatory

**3-** errors.rejectValue(“username”,”errors.legnth”,new Object[] {“Username”,”5”,”9”} ,”Default value”) -> Display Username length must be b/w 5,9

**4-** errors.rejectValue(“password”,”errors.legnth”,new Object[] {“password”,”4”,”8”} ,”Default value”) -> Display Password length must be b/w 4,8

**In JSP**

<form:error path=”username”/>

<form:error path=”password”/>

**Q-66: Explain about the Command Class & How to implement validation in Spring Framework**

**A:1-**Command is mainly used to store client submitted data

**2-** Command class is simple java Bean Class with Private Variables and Public setter and getter methods.

**3-**Client submitted data will be stored in command object by calling setter methods

**4-**Data in the command object will be collected by calling getter methods and will be populated into JSP form elements.

**Command Validation**

Write the Validations required for your Command data in separate Validator class with the following steps

**1-**Write your Validator class by implementing Validator interface which is available in org.springframework.validation package

**2-**Overridethe following two methods

1. Boolean supports(Class<?> clazz)
2. Void validate(Object command , Errors errors)

**3-** Write the code inside the supports() method to check whether correct command is used or not

**4-**Write the code inside the validate() method to validate input data

**5-**When any input data is voiliting validation rules then add the error messages using the following methods:

1. void rejectValue(String , String , Object[] , String)
2. void rejectValue(String , String )

**Q-67: Explain the *@Controller* annotation.**

**A:** The @Controller annotation indicates mark the class as Controller class.it is applied on the class.

**Q-68: Explain *@RequestMapping* annotation.**

**A:**@RequestMapping annotation maps the request with the method.it is applied on the method.

**Q-69: Difference between @Component, @Controller, @Repository & @Service annotations?**

**A:@Component->** it is used to indicate that a class is a component.These classes are used for auto detection and configured as bean, when annotation based configuration are used.

**@Controller->** it is a specific type of component , used in MVC applications and mostly used with RequestMapping annotation

**@Repository->** this annotation is used to indicate that a component is used as repository and a mechanism to store/retrieve/search data. We can apply this annotation with DAO pattern implementation classes.

**@Service->** it is used to indicate that a class is a Service Provider class. Usually the business facade that provide some service annotated with this.

**Q-70: Can we have multiple Spring configuration files?**

**A:** YES. You can have multiple spring context files. There are two ways to make spring read and configure them.

**a) Specify all files in web.xml file using contextConfigLocation init parameter.**

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

WEB-INF/spring-dao-hibernate.xml,

WEB-INF/spring-services.xml,

WEB-INF/spring-security.xml

</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

**b) OR, you can import them into existing configuration file you have already configured.**

<beans>

<import resource="spring-dao-hibernate.xml"/>

<import resource="spring-services.xml"/>

<import resource="spring-security.xml"/>

... //Other configuration stuff

</beans>

**Q-71: Difference b/w <context:annotation-config> vs <context:component-scan>**

**A:** 1) First big difference between both tags is that <context:annotation-config> is used to activate applied annotations in already registered beans in application context. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.

2) Second difference is driven from first difference itself. It registers the beans defined in config file into context + it also scans the annotations inside beans and activate them. So <context:component-scan> does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

**<context:annotation-config>** = Scanning and activating annotations in “already registered beans”.

**<context:component-scan>** = Bean Registration + Scanning and activating annotations

**Q-72: Does Spring MVC provide validation support.**

**A:** yes

**Q-73: How does Spring MVC provide validation support?**

**A:** Spring supports validations primarily into two ways.

Using **JSR-303 Annotations** and any reference implementation e.g. Hibernate Validator

Using custom implementation of **org.springframework.validation.Validator interface**

**Q-74:How to validate form data in Spring Web MVC Framework?**

**A:**Spring MVC supports validation by means of a validator object that implements the **Validator interface**. You need to create a class and implement Validator interface. In this custom validator class, you use utility methods such as rejectIfEmptyOrWhitespace() and rejectIfEmpty() in the ValidationUtils class to validate the required form fields.

@Component

public class EmployeeValidator implements Validator

{

public boolean supports(Class clazz) {

return EmployeeVO.class.isAssignableFrom(clazz);

}

public void validate(Object target, Errors errors)

{

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "error.firstName", "First name is required.");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "error.lastName", "Last name is required.");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "error.email", "Email is required.");

}

}

**Q-75: How to handle exception in Spring MVC Framework**

**A:**Spring MVC framework provide following ways to help us achieving robust exception handling.

**1-Controller Based->** We can define exception handler methods in our controller classes. All we need is to annotate these methods with **@ExceptionHandler**  annotation

**2-Global Exception Handler->** Exception Handling is a cross cutting concern and spring provides **@ControllerAdvice** annotation that we can use with any class to define our global exception handler.

**3-HandlerExceptionResolver implementation->** For generic exceptions,most of the times we serve static pages.Spring Framework provides **HandlerExceptionResolver**

Interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provide default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

**Q-76: How to implement Spring Security.**

**Q-77: what is loose coupling, how to achieve it**

**A:** 1. In Loose-Coupling, when one object is depending on another class object, some external entity will provide that dependency object to the main object that external object we call as a Container.  
2.  In order to get loose-coupling between objects the following two rules are required  
 **a- >** The classes should follow POJI/POJO model.  
 **b->**  Apply dependency injection mechanism.

**1-class Traveler**

{

Vehicle v;

public void setV(Vehicle v)

{

this.v = v;

}

void startJourney()

{

v.move();

}

}

**2-Interface Vehicle**

{

void move();

}

**3-class Car implements Vehicle**

{

public void move()

{

// logic

}

}

**4-class Bike implements Vehicle**

{

public void move()

{

// logic

}

}

3.    In the above traveler class, an external entity injects either car (or) Bike object.  
4.    In traveler, these are no changes required we are shifting the dependency from car to a Bike.  
5.    In the above traveler class, we are token vehicle reference, so that an external object (Container) can injects either car object (or) Bike object, depends on requirement if a traveler.

**Q-78: can we achieve loose coupling using classes**

**A:** We can achieve loose coupling by interface.

In above example, spring container will inject either Car object or Bike object into the Traveler by calling setter method, So if Car object is replaced with Bike then no changes are required in Traveler class, this means there is loose coupling between Traveler and Vehicle object.