Simple Object Access Protocol (SOAP) and Representational State Transfer (REST)

**Java Beans:**

\* Implements java.io.Serializable interface   
\* Provides no argument constructor   
\* Provides getter and setter methods for accessing it's properties.

**JSP:**

[JSP is a webpage scripting language](http://www.withoutbook.com/Technology.php?tech=2&subject=JSP%20Interview%20Questions%20and%20Answers) that can generate dynamic content.

JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.

It’s easier to code in JSP than in Java Servlets.

In MVC, jsp act as a view.

JSP are generally preferred when there is not much processing of data required.

The advantage of JSP programming over servlets is that we can build [custom tags](http://www.withoutbook.com/InterviewQuestionAnswer.php?tech=2&quesId=458&subject=JSP%20Interview%20Questions%20and%20Answers) which can directly call [Java beans](http://www.withoutbook.com/Technology.php?tech=35&subject=Java%20Beans%20Interview%20Questions%20and%20Answers).

We can achieve functionality of JSP at client side by running [JavaScript](http://www.withoutbook.com/Technology.php?tech=8&subject=JavaScript%20Interview%20Questions%20and%20Answers) at client side.

**Servlets:**

[Servlets are Java programs](http://www.withoutbook.com/Technology.php?tech=3&subject=Servlets%20Interview%20Questions%20and%20Answers) that are already compiled which also creates dynamic web content.

Servlets run faster compared to JSP.

Its little much more code to write here.

In MVC, servlet act as a controller.

servlets are best for use when there is more processing and manipulation involved.

There is no such custom tag facility in servlets.

There are no such methods for servlets.

**Lisitners in Servlet:**

ServletContextListener, ServletContextAttributeListener, HttpSessionListener and ServletRequestListener

**Cookies:**

Cookies are used a lot in server client communication. Servlet API HttpSession uses cookie for session management.

Servlet Cookie class and how we can use it to add cookies in response and get cookie details from request object

**Upload a file using Servlet:**

1.Method should be POST.

2. enctype of form should be **multipart/form-data**.

3. To select a file from user file system, we need to use **input** element with **type** as **file**.

<form action="UploadDownloadFileServlet" **method="post"** **enctype="multipart/form-data">**

Select File to Upload:<**input type="file"** name="fileName">

**SpringAdvantages:**

Spring provides a layered architecture.

Spring Provide pojo programming

Pojo programming enables continuous integration and testablility

Dependency injection And Inversion of control simplifies the JDBC

**Inversion OF Controller (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 Orient programming(AOP):**

Aspect oriented (AOP): Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.

**Container:**

Spring contains and manages the life cycle and configuration of application objects.

**MVC Framework:**

Spring comes with MVC web application framework, built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. But other frameworks can be easily used instead of Spring MVC Framework.

**Transaction Management:**

Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. **Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.**

**JDBC Exception Handling:**

The JDBC abstraction layer of the Spring offers a meaningful exception hierarchy, which simplifies the error handling

**You need not handle any database-related exceptions explicitly instead Spring JDBC Framework will handle it for you. All the exceptions thrown by the Spring JDBC Framework are subclasses of *DataAccessException* which is a type of RuntimeException, so you need not handle it explicitly. Any checked exception when thrown will be mapped to any of the subclasses of the *DataAccessException*by the framework.**

**Don't handle any exceptions in the DAO layer, instead throw it to the front-end and handle it. To handle the exception all you need to do is to catch the appropriate exception in the hierarchy and address it.**

package com.vaannila.dao;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.dao.DataAccessException;

import org.springframework.dao.DuplicateKeyException;

import org.springframework.dao.EmptyResultDataAccessException;

import com.vaannila.domain.Forum;

public class Main {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(

"beans.xml");

ForumDAO forumDAO = (ForumDAO) context.getBean("forumDAO");

Forum springForum = new Forum(1, "Spring Forum",

"Discuss everything related to Spring");

try {

forumDAO.insertForum(springForum);

} catch (DuplicateKeyException e) {

System.out.println("Forum Already Exist");

} catch (DataAccessException e) {

e.printStackTrace();

}

try {

System.out.println(forumDAO.selectForum(2));

} catch (EmptyResultDataAccessException e) {

System.out.println("The Forum id is invalid");

} catch (DataAccessException e) {

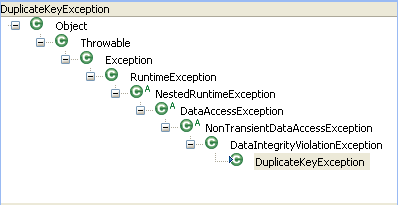
e.printStackTrace();

}

}

}

 When you run this example for the first time a Forum will be created, when you run it again you will get a **DuplicateKeyException**, which we catch it in the catch block and display the appropriate message. Only one Forum exist in the database with the forum id '1', if you query for the forum with forum id '2' then you will get the "The Forum id is invalid" message dispalyed on the console.



**The core container:** The core container provides the essential functionality of the Spring framework. A primary component of the core container is the BeanFactory, an implementation of the Factory pattern. The BeanFactory applies the Inversion of Control (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.

**Spring context:** The Spring context is a configuration file that provides context information to the Spring framework. The Spring context includes enterprise services such as JNDI, EJB, e-mail, internalization, validation, and scheduling functionality.

**Spring AOP:** The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result you can easily AOP-enable any object managed by the Spring framework. The Spring AOP module provides transaction management services for objects in any Spring-based application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

**Setter Injection:** Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

**Constructor Injection:** Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

**BeanFactory:**

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.   
BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.   
BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

**ApplicationContext:**

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the **application context**. On the surface, an **application context** is same as a **bean factory**. Both load bean definitions, wire beans together, and dispense beans upon request.

**ResourceLoader support:** Spring�s Resource interface us a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader, Hence provides an application with access to deployment-specific Resource instances.

**BeanFactory:**

Bean life cycle in Spring Bean Factory Container is as follows:   
  
 The spring container finds the bean�s definition from the XML file and instantiates the bean.   
 Using the dependency injection, spring populates all of the properties as specified in the bean definition   
 If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean�s ID.   
 If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.   
 If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called.   
 If an init-method is specified for the bean, it will be called.   
 Finally, if there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.

**BeanWiring:**

The act of creating associations between application components (beans) within the Spring container is reffered to as Bean wiring.

**AutoWiring:**

The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. The autowiring functionality has five modes.

►no ► byName ► byType ► constructor► autodetect

**SpringModules:**

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

**CoreContainerModule:**

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.

**ApplicationContextModule:**

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.

**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.

**Spring JDBC 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.

**SpringORM:**

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provide support to tie into several popular ORM frameworks, including Hibernate, JDO, and iBATIS SQL Maps. Spring's transaction management supports each of these ORM frameworks as well as JDBC

**Spring AOP Topics:**

**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.

**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.

**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.

**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.

**Proxy:**

 When you think of client objects the target object and the proxy object are the same.

**IOC Briefly:**

**As the name implies Inversion of control means now we have inverted the control of creating the object from our own using new operator to container or framework. Now it’s the responsibility of container to create object as required. We maintain one xml file where we configure our components, services, all the classes and their property. We just need to mention which service is needed by which component and container will create the object for us. This concept is known as dependency injection because all object dependency (resources) is injected into it by framework.  
  
Example:  
<bean id="createNewStock" class="springexample.stockMarket.CreateNewStockAccont">   
<property name="newBid"/>  
</bean>  
In this example CreateNewStockAccont class contain getter and setter for newBid and container will instantiate newBid and set the value automatically when it is used. This whole process is also called wiring in Spring and by using annotation it can be done automatically by Spring, refereed as auto-wiring of bean in Spring.**