## Spring Framework

**Q)** I want to develop the High performance large Scale distributed enterprise application. As J2EE Architect (25L), suggest the best Technologies and frameworks for application architecture.

| SERVICES | Technologies & Frameworks (Without Spring) | INTRODUCED IN SPRING | TO INTIGRATE WITH |
|---|---|---|---|
| Web Support | Servlets, JSP, Struts, Jsf, Struts2, Flex | Spring MVC | All |
| Persistence Service | JDBC, Hibe3rnate, JPA, Entity Bean (EJB) | - | All |
| Transaction Service | JDBC Transaction, Hibernate Transaction, JPA Transaction, EJB Container Transaction | Spring Transaction | - |
| Security Service | JAAS, Third Party | Spring Security | All |
| Remoting Service | RMI, EJB 2/3, Web services | - | All |
| Timer Service | JDK, EJB Container | - | - |
| Messaging service | JMS, MDB | - | All |
| Resource Management | Web Container, EJB Container | Spring Container | - |
| Life Cycle Management | Web Container, EJB Container | Spring Container | - |
| Mailing Service | Java Mail API | - | All |
| Registery Service | RMI Registry, JNDI Registry, UDDI | - | All |
| Logging Service | Log4j | - | - |
| Unit Testing | JUnit, Mock objects, Easy mock | - | All |
| IOC | EJB3 Container | Spring Container | - |
| AOP | EJB3 container | Spring Container | - |
| Mobile Support | J2ME, Iphone, Android, Windows phone 7 etc | - | Spring integration With android |
| Big data Support | Hadoop | - | Spring Integration With Hadoop |

**Spring Framework Modules**

1. Spring IOC(*)
2. Spring AOP (*)
3. Spring Data Access
    a) Spring DAO Support.
    b) Spring with JDBC
    c) Spring with Hibernate
    d) Spring with JPA
    e) Spring with IBatis
4. Spring Transaction Management
5. Spring Web
    a) Spring with struts1
    b) Spring with Struts2
    c) Spring With JSF
6. Spring web MVC (*)
7. Spring security (*)
8. Spring Integrations
    a. Spring with java Mail
    b. Spring with RMI
    c. Spring with EJB2
    d. Spring with EJB3
    e. Spring with JAXB. (oxm)
    f. Spring with web Services. (JAX-WS, JAX-RS)
9. Spring EL.

**Spring Other Projects**

1. Spring Mobile
2. Spring Roo
3. Spring Batch
4. Spring Data
5. Spring Social
6. Spring web Flow
7. Spring Blaze-DS (Flex).
8. Spring Android.
9. Spring Hadoop

**Spring IOC (Inversion of Control)**

- When you are developing any component or bean, it may contain some dependencies.
- Those Bean dependencies can be initialized in two ways.
    Create the object and initialize.
    Lookup the registry and initialize.
- With Spring IOC or DI (Dependency Injection), Spring Container is responsible for initializing or Injecting Bean dependencies automatically.

| Without Spring IOC | With Spring IOC |
|---|---|
| Class A{<br>Void m1(){….}<br>} | Class A{<br>Void m1(){….}<br>} |

| | |
|---|---|
| Class B{<br>Void m2(){….}<br>}<br><br>Class Hello{<br>A aobj=null;<br>B bobj=null;<br><br>Public Hello(){<br>Aobj=new A();<br>Bobj=ctx.lookup(“B-JNDI”);<br>}<br><br>Void Show(){<br>Aobj.m1();<br>Bobj.m2();<br>}<br>} | Class B{<br>Void m2(){….}<br>}<br><br>Class Hello{<br>@Autowired<br>A aobj=null;<br><br>@Resource<br>B bobj=null;<br><br>Void show(){<br>Aobj.m1();<br>Bobj.m2();<br>}<br>} |

Note:

Every class you are writing in spring application can be called as Spring Bean.

- Hello Bean needs A object and B object.
- Hello Bean needs A resource and B resource.
- Hello Bean needs A Bean and B Bean

- Hello Bean is depending on two beans called A bean and B bean.
- Hello Bean dependencies called A and B has to be Initialized.
- Hello Bean dependencies called A and B has to be Injected
- With Spring IOC, Spring container Injects the Hello Bean dependencies called A and B.
- Spring Dependency injection uses 3 ways to inject the dependencies
    1. Setter Injection
    2. Constructor Injection
    3. Field Injection. (using Annotations)

Ex:

```
Class Student{
        String sid;                 //1 setter Injection
        String sname;    //2 Constructor Injection
        @Autowired
        Address add;     //3 Field Injection. (using Annotations)
        public void setSid(String sid) {
        This.sid=sid;               //1 Setter Injection
        }
        Public Student(String sname){
        This.sname=sname; //2 Constructor Injection
        }
}
```

**First Spring Example setup in Eclipse**

1. Create the java project with the name: Jtc1.

2. Add all the Spring3 jars (26 JARS) to project build path.
3. Copy Spring Configuration Document to src folder.
   jtcindia.xml (default is application Context.Xml)
4. Create a package called com. Jtcindia. Spring and write the following:
   a. A.java
   b. B.java
   c. Hello.java
   d. Jtc1.java
5. Update the Spring Configuration Document.
   a. Jtcindia.xml

**Jtc 1: Files required**

| | |
|---|---|
| Jtc1. Java | A.java |
| B.java | Hello.java |
| Jtcindia.xml | |

Jtc1. Java

```
Package com.jtcindia.spring;
Import org.springframework. context.
Application Context;
Import org.springframework. context. Support. classPathxmlApp
lication
Context;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ see          : www.jtcindia.org
**/
Public class Jtc1{
Public static void main(String[] args){
/*
//without IOC
A aobj=new A();
Aobj.setA(99);
Aobj.setMsg("HelloGuys");
B bobj=new B(88,"Hai Guys");
Hello hello=new Hello(bobj);
Hello.setAobj(aobj);
// hello.show();
*/

ApplicationContext ctx=new
Class pathxmlApplicationContext ("jtcindia.xml");
System.out.println("spring Contianer is Ready");
System.out.printIn("----------);
```

A.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ see          : www.jtcindia.org
**/
Public class A{
Private int a;          //S.I

Private String msg;    //S.I

Public A(){
System.out.printIn("A-D.C");
}

Public void seta(int a) {
System.out.printIn("A-setA()");
This.a=a;
}

Public void setMsg(String msg) {
System.out.println("A-setMsg()");
This.msg=msg.
}

Public void showA(){
System.out.printIn("A-showA()");
System.out.printIn(a);
```

```
Hello h=(Hello)ctx.getBean("hello");
h.show();
}
```

```
System.out.printIn(msg);
}
}
```

B.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ see          : www.jtcindia.org
**/
Public class B{
Private in b;          //c.l
Private string str;  //c.l
Public B(int b, String str) {
System.out.printIn("B-2 arg");
This.b=b;
This.str=str;
]
Public void showB(){
System.out.printIn("B-showmB()");
System.out.println(b);
System.out.println(str);
}
}
```

Hello.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ see          : www.jtcindia.org
**/
Public class Hello{
Private A aobj;                ///s.l
Private B bobj;                //c.l
Public void setAobj(A aobj) {
System.out.println("Hello-setAobj");
This.aobj=aobj;
}
Public Hello(B bobj) {
System.out.println("Hello-1 arg");
This.bobj=bobj;
}

Public void show(){
Aobj.showA();
Bobj.showB();
} }
```

Jtcindia.xml

```
<?xml version="1.o"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="aobj" class="com.jtcindia.spring.A">
<property name="a" value="99"/>
<property name="msg"value="Hello Guys"/>
</bean>
<bean id="bobj" class="com.jtcindia.spring.B">
<constructor-arg value="88"/>
<constructor-arg value="Hai Guys"/>
</bean>
<bean id="hello" class="com.jtcindia.spring.Hello">
<property name="aobj" ref="aobj"/>
<constructor-arg ref="bobj"/>
```

```
</bean>
</beans>
```

Here, Spring container is doing

1. Spring container is creating the resources.
2. Spring container is initializing the resources.
3. Spring container is using the Single-ton pattern.
4. Spring container is using the Factory pattern.

Q1) What is a Spring Bean?
Ans: Every class You are writing in spring application can be called as Spring Bean

Q2) What is Dependency?
Ans: Any property available in a Bean class which needs to be initialized is called as Dependency.

Q3) What is Injection?
Ans: Initialization of Bean class property is also called as Injection.

Q4) What is Dependency Injection (DI) ?
Ans: The process of initializing Bean Dependencies is called as Dependency Injection.

Q5) What are types of Dependency Injection supported by Spring?
Ans: Spring Dependency Injection uses 3 ways to inject the dependencies

1. Setter Injection
2. Constructor Injection
3. Field Injection. (using Annotations)

Q6) What is Setter Injection?
Ans: The process of initializing Bean Dependencies with setter methods is called as setter Injection.
Q7) How can I implement Setter Injection?
Ans: Using <property> tag.

Q8) What is Constructor Injection?
Ans: The process of initializing Bean Dependencies with Constructor is called as Constructor injection.

Q9) How can I implement Constructor Injection?
Ans: Using <constructor-arg> tag.

Q10) what is Bean Definition?
Ans: Configuring Bean and its dependencies in Spring Configuration Document is called as Bean Definition.

Q11) When should I use <property> tag and <constructor-arg> tag?
Ans:

Q12) when should I use value attribute and ref attribute?
Ans: Use value attribute for specifying values for simple type of variables like
Primitive Variables
wrapper Variables
String Variables

Use ref attribute for specifying bean references for reference type variables.

Q13) What are the GOF Patterns used in Spring IOC container?
Ans: Single-ton pattern and Factory pattern.

Q14) What is spring Container?
Ans: ApplicationContext object is called as Spring Container.

Q15) What will be done by the spring Container when it finds following bean definition?

```
<bean id="aobj" class="com.jtcindia.spring.A">
<property name="a"value="99"/>
<property name="msg" value="Hello Guys"/>
</bean>
```

Ans:

```
A aobj=bew A ();
Aobj.setA(99);
Aobj.setMsg("HelloGuys");
```

Q16) What will be done by the spring container when it finds following bean definition?

```
<bean id="bobj" class="com.jtcindia.spring.B">
<constructor-arg value="88"/>
<constructor-arg value="Hai Guys"/>
</bean>
```

Ans:

```
B bobj=new B(88,"Hai Guys");
```

Q17) What will be done by the spring Container when it finds following bean definition?

```
<bean id="hello" class="com.jtcindia.spring.Hello">
<property name="aobj" ref="aobj"/>
<constructor-arg ref="bobj"/>
</bean>
```

Ans:

```
Hello hello=new Hello(bobj);
Hello.setAobj(aobj);
```

Q18) What is spring Configuration file?
Ans: spring Configuration file is an XML file which contains various bean definitions.

Q19) what will happen with the following code?

```
<beans>
<bean id="hello" class="com./jtc.hello'/?>
</kbeans>
```

Hello hello=(Hello) ctx.getbean("hello1");

Ans: Exception will be raised
NoSuchBeanDefinitionException: No Bean named 'hello1' is defined
Q20) Can I configure two beans with same Id?
<beans>
<bean id="hello"class="com.jtc.Hello1"/>
<bean id="hello"class="com.jtc.Hello2"/>
</beans>

Ans: No, Bean Id must be unique.
There are multiple occurrences of Id value 'hello'

Q21) Can I configure two beans with same Bean class?
Ans: Yes.
i.e

```
<beans>
<bean id="hello1" class="com.jtc.Hello"/>
<bean id="hello2" class=" com.jtc.Hello"/>
</beans>

Hello h1=(Hello) ctx.getBean("hello1");
Hello h2=(Hello) ctx.getBean("hello2");
Here h1 !=h2
```

Q22) What will happen with the following code?

```
<beans>
<bean id="hello" class=" com.jtc.Hello">
<constructor-arg value="99"/>
</bean>
</beans>
```

```
Class Hello{
Int x;
// No constructor
}
```

Ans. BeanCreationException:Error creating bean with name 'hello' defined in class path resource [jtcindia.xml]: could not resolve matching constructor

Q23) What will happen with the following code?

```
<beans>
<bean id="hello" class="com.jtc.Hello">
<property name="x" value ="99"/>
</bean>
</beans>
```

```
Class Hello{
Int x;
// No setter
}
```

Ans: Not Writable Property Exception: Invalid property 'x' of bean class [A]: bean property 'x' is not writable or has an invalid setter method.
Does the parameter type of the setter match the return type of the getter?

**Bean Scopes**

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:**

- When bean scope is singleton then only one instance will be created for that bean and the same instance will be returned when you call getBean () method.
- Singleton is the default scope in the Application Context container.
- When scope is single-ton then default loading type is aggressive loading.

**2) prototype**

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

**3) request:**

- Request scope is equals to HttpServletRequest in the web application.

**4) session:**

- Session scope is equals to HttpSession in the web application**.**

**5) global-session**

- Global-session scope is equals to session in the portlet based web application**.**

**Bean Loading Types**

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

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

**Q24) How many bean scopes are there?**
**Ans:** 5

**Q25) what is default bean scope?**
**Ans:** singleton

**Q26) what is the difference between singleton and prototype?**
**Ans;**

**Q27) How many bean loading types are there?**
**Ans: 2**

**Q28) What is default bean loading type?**
**Ans:**

**Q29) What is the difference between Lazy loading and Aggressive loading?**
**Ans:**

**Answer the following questions based on Jtc1.**
Q30) what will happen with the following?

```
<bean id="ao"class="….A">
….
</bean>
<bean id="bo" class="….B">
….
</bean>
<bean id="hello" class=" ….Hello">
….
</bean>
```

**Ans:** All 3 beans will be loaded at container start-up and all are singletons.
Order of loading: A-B-Hello

**Q31) What will happen with the following?**

```
<bean id="ao" class="….A" lazy-init="true">
….
```

```
< /bean>
<bean id=”bo” class=”….B” lazy-init=”true”>
….
</bean>
<bean id=”hello” class=”….Hello” lazy-init=”true”>
….
</bean>
```

**Ans:** No bean will be loaded at container start-up,
All are singleton
Loads When you call get Bean () method
Order of loading: B, Hello, A

**Q32)** What will happen with the following?

```
<bean id=”ao” class=”…..A” lazy-init=”true”>
….
</bean>
<bean id=”bo” class=”….B” lazy-init=”true”>
….
</bean>
<bean id=”hello” class=”….Hello” lazy-init=”false”>
….
</bean>
```

Ans: All 3 Beans will be loaded at container start-up and all are singletons.
Order of loading: B-Hello – A

Q33) What will Happen with the following?

```
<bean id=”ao” class=”….A” lazy-init=”false”>
….
</bean>
<bean id=”bo” class=”….B” lazy-init=”false”>
….
</bean>
<bean id=”hello” class=”….Hello” lazy-init=”true”>
….
</bean>
```

Ans: 2 beans will be loaded at container start-up and all are singletons.
Order of loading: A-B

**Q34**) What will happen with the following?

```
<bean id=”ao” class=”….A” lazy-init=”false” scope=”prototype”>
….
```

```
</bean>
<bean id="bo" class="….B" lazy-init="false" scope="prototype">
….
</bean>
<bean id="hello" class="….Hello" lazy-init="true" scope="singletor">
….
</bean>
```

Ans: No beans will be loaded at container start-up.
A, B are prototype.
Hello is singleton.

Q35) What will happen with the following?

```
<bean id="ao" class="….A" scope="prototype">
….
</bean>
<bean id="bo" class="….B" scope="prototype">
….
</bean>
<bean id="hello" class="….Hello" scope="singleton">
….
</bean>
```

Ans. All 3 beans will be loaded at container start-up
A, B are prototype.
Hello is singleton.
Order of loading: B- Hello – A