Injecting various types of Bean Properties

Bean may contain the properties of following types.

1. Simple Types (primitives, wrappers, strings and Dates)
2. List of simple types
3. Set of simple types
4. Map of simple types
5. Properties of simple types
6. Other Bean types
7. Collection of other Bean types

The above said properties can be injected using setter injection or constructor injection.

1. Simple Types:

```
Class customer{
        Int cid;
        String cname;
        Long phone;
}
```

Constructor Injection	Setter injection
<bean id=”cust” Class=”com.jtcindia.spring.customer”> <constructor-arg value=”101”/> <constructor-arg value=”Som”/> <constructor-arg value=”9999”/> </bean>	<bean id=”cust1”class=” com.jtcindia.spring. Customer”> <property name=”cid” value=”102”/> <property name=”cname” value=”Prakash”> <property name=”phone” value=”8888”/> </bean>

2) List of simple types

```
Class Customer{
        List<String> emails;
}
```

Constructor Injection	Setter Injection
<bean id=”cust” class=”com.jtcindia. spring.customer”> <constructor-arg> <list> <value>aa@jtc</value> <value>bb@jtc</value> </list> </constructor-arg> </bean>	<bean id=”cust” class=”com.jtcindia. spring.Customer”> <property name=”emails”> <list> <value>aa@jtc</value> <value>bb@jtc</value> </list> </property> </bean>

3) Set of Simple types

```
Class customer{
    Set<string> emails;
}
```

Constructor Injection	Setter Injection
<bean id=”cust” class=”com.jtcindia. spring.customer”> <constructor-arg> <set> <value>aa@jtc</value> <value>bb@jtc</value> </set> </constructor-arg> </bean>	<bean id=”cust” class=”com.jtcindia. spring.Customer”> <property name=”emails”> <set> <value>aa@jtc</value> <value>bb@jtc</value> </set> </property> </bean>

4. Map of simple types

```
Class customer{
    Map<string,long> refs;
}
```

Constructor Injection	Setter Injection
<bean id=”cust” class=”com.jtcindia. spring.customer”> <constructor-arg> <map> <entry key=”AA” value=”11”/> <entry key=”BB” value=”22”/> </map> </constructor-arg> </bean>	<bean id=”cust” class=”com.jtcindia. spring.Customer”> <property name=”refs”> <map> <entry key=”AA” value=”11”/> <entry key=”BB” value=”22”/> </map> </property> </bean>

5) properties of Simple types

```
Class Customer{
    Properties Myprops;
}
```

Constructor Injection	Setter Injection
<bean id=”cust” class=”com.jtcindia. spring.customer”> <constructor-arg> <props> <entry key=”AA” >11</prop> <entry key=”BB” >22</prop> </props> </constructor-arg> </bean>	<bean id=”cust” class=”com.jtcindia. spring.Customer”> <property name=”myprops”> <props> <entry key=”AA” >11</prop> <entry key=”BB” >22</prop> </props> </property> </bean>

6) Other Bean types

```
Class Customer{
        Address address;
}
```

Constructor Injection	Setter Injection
<bean id=”add” class=” com.jtcindia. spring. Address”/> <bean id =”cust” class=”com.jtcindia.spring.Customer”> <constructor-arg ref=”add”/> </bean>	<bean id=”add” class=”com. Jtcindia.spring.Address”/> <bean id=”cust” class=”com.jtcindia. spring. Customer”> <property name=”address” ref=”add”/> </bean>

7) collection of other Bean types

```
Class Customer{
        List<Address> accounts;
}
```

Constructor Injection	**Setter Injection**
<bean id=”add1” class=”com.jtcindia.spring.Address”/> < bean id=”add2” class=”com.jtcindia. spring.Address”/> <bean id=”cust” class=”com.jtcindia.spring.customer”> <constructor-arg> <list> <ref bean=”add1”/> <ref bean=”add2”/> </list> </constructor-arg>	<bean id=”add1” class=”com.jtcindia.spring.Address”/> < bean id=”add2” class=”com.jtcindia. spring.Address”/> <bean id=”cust” class=”com.jtcindia.spring.customer”> <property name=”addresses”> <list> <ref bean=”add1”/> <ref bean=”add2”/> </list> </property>

</bean>	</bean>

Jtc 2: Files Required

Jtc2.java	Account.java
Address.java	Customer.java
Jtcindia.xml	

Jtc 2.java

```
Package com.jtc india.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc2{
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext("jtcindia.xml");
Customer cust=(Customer)ctx.getBean("cust");
Cust.show();
}
}
```

Account.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Account {
Private int accno;
Private string atype;
Private double bal;
Private Account(){
System.out.println("Account-D.C");
}
// Setters and Getters
Public String toString(){
Return""+accno+"\t"+atype="\t"+bal;
}}
```

Address.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Account {
Private String city;        private String street;
Private String state;
Public Address(String city, string street, string
state){
System.out.println("Address-3 arg");
This.city=city;             this.street=street;
This.state=state;
}
Public string to String(){
Return ""+city+"\t"+street+"\t"+state;
}}
```

Customer.java

```
Package com.jtcindia.spring;
Import java.util.*;
Public class customer {
Private int cid;
Private string cname;
Private string email;
Private long phone;
Private list<String> emails;
Private set<Long> phones;
Private Map<string,long> refs;
Private properties myprops;
Private address address;
Private list<Account> accounts;
Public customer (int cid, string cname, string email,
Long phone){
System.out.println("customer 4 arg");
This.cid=cid;
This.cname=cname;
This.email=email;
This.phone=phone;
}
// Setters and Getters

Public void show(){
System.out.println(cid);
System.out.println(cname);
System.out.println(email);
System.out.println(phone);
System.out.println(emails);
System.out.println(phones);
System.out.println(refs);
System.out.println(myprops);
System.out.println(address);
For(Account acc:accounts){
System.out.printIn(acc);
}
System.out.println(emails.getClass().getName());
System.out.println(phones.getClass().getName());
System.out.println(refs.getClass().getName());
System.out.println(myprops.getClass().getName());
}
}
```

Jtcindia.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="add"class="com.jtcindia.spring.Address">
<constructor-arg value="Noida"/>
<constructor-arg value="SECTOR2"/>
<constructor-arg value="UP"/>
</bean>
<bean id="acc1" class="com.jtcindia.spring.Account">
<property name="accno"value="11"/>
<property name="atype"value="SA"/>
<property name="bal" value="10000.0"/>
</bean>
<bean id="acc2" class="com.jtcindia.spring.Account">
<property name="accno"value="12"/>
<property name="atype" value="SA"/>
<property name="bal" value="20000.0">
</bean>
```

```
<bean id="acc3" class=" com.jtcindia.spring.Account">
```

```
<property name=”accno” value=”13”/>
<property name=”atype” value=”CA”/>
<property name=”bal” value=” 350000.0”/>
</bean>
<bean id=”cust” class=”com.jtcindia.spring.customer”>
<constructor-arg value=”101”/>
<constructor-arg value=”Som”/>
<constructor-arg value=”som@jtc”/>
<constructor-arg value=”9999”/>
<property name=”emails”>
<list>
<value>aa@jtc</value>
<value>bb@jtc</value>
<value>cc@jtc</value>
</list>
</property>
<property name=”phones”>
<set>
<value>111</value>
<value>222</value>
<value>333</value>
</set>
</property>
<property name=”refs”>
<map>
<entery key=”AA” value=”11”/>
<entery key=”BB” value=”22”/>
<entery key=”CC” value=”33”/>
</map>
</property>
<property name=”myprops”>
<props>
<prop key=”AA”>11</prop>
<prop key=”BB”>22</prop>
<prop key=”CC”>33</prop>
</props>
</property>
<property name=”address” ref=”add”/>
<property name=”accounts”>
<list>
<ref bean=”acc1”/>
<ref bean=”acc2”/>
<ref bean=”acc3”/>
</list>
</property>
</bean>
</beans>
```

Wiring

- Wiring is the process of injecting the Dependencies of the Bean.
- Wiring can be done in two ways.
    1. Explicit wiring
    2. Implicit wiring or auto wiring

Explicit 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>
```

Implicit wiring or Auto wiring:

- In the case of Auto wiring, spring Container can detect the Bean Dependencies automatically and injects those Dependencies.

```
<beans…>
        <bean id=”ao” class=”…A”/>
        <bean id=”bo” class=”…B”/>
        <bean id=”hello” class=”…Hello” autowire=”xxx”/>
</beans>
```

Following are possible values for autowire attribute

1. byname
2. byType
3. constructor
4. autodetect (Not in 3.0)

byname:

- When autowire attribute value 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.
- When bean is found with the matching name then it will be injected otherwise bean property remains uninjected.
- Bean will be instantiated using Default constructor,
- Dependent Bean Instances will be detected using bean Name.
- Detected bean instances will be injected through setter injection only.

Jtc3: Files required

Jtc3.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtc3.java

```
Package com.jtcindia.spring;
Import org.springframework.context.ApplicationContext;
Import org.springframework.context.cupport.classPathXmlApplicationContext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc3 {
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext("jtcindia.xml");
Hello hello=(Hello)ctx.getBean("hello");
Hello.show();
} }
```

A.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class A {
Private int a;
Private string msg;
Public void seta(int a) {
This.a=a;
}
Public void setMsg(String msg) {
This.msg=msg;
}
Public String to String(){
Return""+a+"\"+msg;
}
}
```

B.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class B {
Private int b;
Private string str;
Public B(int b, String str) {
This.b=b
This.str=str;
}
Public String to String(){
Return""+b+"\t"+str;
}
}
```

A.java	Jtcindia.xml

Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit : www.jtcindia.org
**/
Public class Hello {
Private A aobj;
Public void setAobj(A aobj){
System.out.println("Hello-setAobj()");
This.aobj=aobj;
}
Public void setBobj(B bobj) {
System.out.println("Hello-setBobj()");
This.bobj=bobj;
}
Public void show(){
System.ouut.println(aobj);
System.ouut.println(bobj);
}
}
	<?xml version="1.0" encoding="UTF-8"?>
	<beans…>
	<bean id="aobj" class="com.jtcindia.spring.A">
	<property name="a" value="10"/>
	<property name="msg" value="AAA"/>
	</bean>
	<bean id="bo" class="com.jtcindia.spring.B">
	<constructor-arg value="20"/>
	<constructor-arg value="BBB"/>
	</bean>
	<bean id="hello" class="com.jtcindia.spring.Hello" Autowire="byname"/>
	</beans>

Jtc 3 Container Start-up Steps:

1. Loads A Bean.
2. A Bean instance will be created by calling default constructor.
3. Calls seta() method to inject the value 10 into property a.
4. Calls setMsg() method to inject the value AAA into pronerty msg.
5. Loads B Bean.
6. B Bean instrance will be created by calling 2 arg constructor.
7. Loads Hello Bean.
8. Hello Bean instance will be created by calling default constructor.
9. Because of byname autowire process, Container will do the following.
    - Checks the dependencies available for Hello Bean
    - Currently two dependencies are available for Hello bean called aobj and bobj.
    - Checks whether any bean is running in the container whose id is same as property name.
    - Matching bean is found for aobj so aobj will be injected with that matching bean by calling setAobj() method.
    - Mathcing bean is not available for bobj so bobj remains uninjected i.e. contains null only.

By Type:

- When autowire attribute value is by Type then spring container checks whether are bean instance running in the container whose Type is same as bean property Date Type or not.
- In this you may get three cases;

1. When exactly one bean is found with the matching data typw then then it will be injected. (Refer Jtc4).
2. When no beans is found with the matching Data Type then bean property not be injected. (refer Jtc5).
3. When two or more beans is found with the matching Data Type then except will be thrown called UnsatisfiedDependencyException

   Error Message;
   Unsatisfied dependency expressed through bean property aobj No unique bean of type (com.jtcindia.spring.A) is defined: expected single matching be but found 2: (ao1, ao2) (Refer Jtc6);

- Bean will be instantiated using Default constructor.
- Dependent Bean Instances will be detected using bean data type.
- Detected bean instances will be injected through setter injection only.

Jtc4: Files required

Jtc4.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtcindia.xml

```
<?xml version=”1.0” encoding=”UTF-8”?>
<beans…>
<bean id=”ao” class=”com.jtcindia.spring.A”>
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”bo” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”hello” class=”com.jtcindia.spring.Hello” Autowire=”byname”/>
</beans>
```

Jtc5: Files required

Jtc5.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtcindia.xml

```
<?xml version=”1.0” encoding=”UTF-8?>
<beans…>
<bean id=”hello”clas=”com.jtcindia.spring.Hello” autowire=” by type”/>
```

```
</beans>
```

Jtc6: Files required

Jtc6.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtcindia.xml

```
<?xml version=”1.0” encoding=”UTF-8?>
<beans…>
<bean id=”ao1” class=”com.jtcindia.spring.A”>
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”ao2” class=”com.jtcindia.spring.A”>
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”bo” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
</bean>
<bean id=”hello” class=”com.jtcindia.spring.Hello”autowire=”byType”/>
</beans>
```

Constructor

- When autowire attribute value is constructor then Spring container cheks whether any bean instance running in the container whose Data Type is same as bean property Data Type or not.,
- Depending on the Availability of Bean Instances, Spring Container identifies the Matching Constructor and invokes that constructor to inject the Bean Dependencies.
- In this you may get three cases;
    1. When no beans is found with the matching Data Type then bean property will not be injected.
    2. When exactly one bean is found with the matching Data Type and without matching constructor then bean property will not be injected.
    3. When exactly on bean is found with the matching Data Type and with matching constructor then it will be injected.
    4. When two or more beans are found with the matching Data Type then
        - Container will try to pick one Bean from available multiple beans based on by Name autowire process first.
        - When one Bean is not selected based on by name autowire process then ignores multiple beans found with given type.

1. Bean will be instantiated using matching constructor.
2. Dependent Bean Instances will be detected using bean data type.
3. Detected bean instances will be injected through constructor injection.

Jtc7: Files required

Jtc7.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello.java

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

Public Hello(A aobj. B bobj) {
System.out.println("Hello-(A,B)");
This.aobj=aobj;
This.bobj=bobj;
}
Public void show(){
System.out.println(aobj);
System.out.println(bobj);
}
}
```

Jtcindia.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans…>
<bean id="hello" class="com.jtcindia.spring.Hello" autowire="constructor"/>
</beans>
```

Jtc8: Files required

Jtc8.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
```

Jtcindia.xml

```
<?xml version="1.0" encoding="UTF-8?>
<beans…>
<bean id="ao" class="com.jtcindia.spring.A">
```

*@Company : java Training Center
*@ visit : www.jtcindia.org
**/
Public class Hello {
Private A aobj:
Private B bobj:
Public Hello(){
System.out.println("Hello-()");
}
Public void show(){
System.out.println(aobj);
System.out.println(bobj);
}
	<property name="a" value="10"/>
	<property name="msg" value="AAA"/>
	</bean>
	<bean id="bo" class="com.jtcindia.spring.B">
	<constructor-arg value="20"/>
	<constructor-arg value="BBB"/>
	</bean>
	<bean id="hello" class="com.jtcindia.spring.Hello" autowire="constructor"/>
	</beans>

Jtc9: Files required

Jtc9.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtcindia.xml

```
<?xml version="1.0" encoding="UTF-8?>
<beans…>
<bean id="ao" class="com.jtcindia.spring.A">
<property name="a" value="10"/>
<property name="msg" value="AAA"/>
</bean>
<bean id="bo" class="com.jtcindia.spring.B">
<constructor-arg value="20"/>
<constructor-arg value="BBB"/>
</bean>
<bean id="hello" class="com.jtcindia.spring.Hello" autowire="constructor"/>
</beans>
```

Jtc10: Files required

Jtc10.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtcindia.xml

```
<?xml version="1.0" encoding="UTF-8?>
<beans…>
<bean id="ao1" class="com.jtcindia.spring.A">
```

```
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”ao2” class=”com.jtcindia.spring.A”>
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”bo” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”bobj” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”hello” class=”com.jtcindia.spring.Hello”autowire=” constructor”/>
</beans>
```

Jtc11: Files required

Jtc11.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtcindia.xml

```
<?xml version=”1.0” encoding=”UTF-8?>
<beans…>
<bean id=”ao” class=”com.jtcindia.spring.A”>
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”aobj” class=”com.jtcindia.spring.A”>
<property name=”a” value=”10”/>
<property name=”msg” value=”AAA”/>
</bean>
<bean id=”bo1” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”bo2” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”hello” class=”com.jtcindia.spring.Hello”
autowire=”constructor”/>
</beans>
```

Autodetect (Not in 3.0)

- Bean Dependencies will be resolved through constructor or byType.

Injecting sub Types into Super Type Reference

- When Spring Container is detecting the beans with byType autowire process, it with find super type and also sub types.

Consider the Case.

```
Class A()
Class B extends A{}
Class Hello{
        A aobj;
        …
}
```

Case 1:

```
<bean id=”ao” class=”…A”/>
<bean id=”hello” class=”…Hello” autowire=”by Type”/>
```

Ans: A instance available in Spring Container will be injected into Hello.

Case 2:

```
<bean id=”bo” class=”…A”/>
<bean id=”bo” class=”…B”/>
<bean id=”hello” class=”…Hello” autowire=”byType”/>
```

Ans:

It throws unsatisfied Dependency Exception.
Two Beans found with the A type one is ao and second is bo.

Consider another case with DAO.

```
Interface customer DAO{}
classJDBCCustomerDAO imp Customer DAO{
        …..
}
Class customerService{
        Customer DAO cdao=null;
        …
}

<bean id=”cdao” class=” =”..JDBCCustomer DAO”/>
<bean id=”cs” class=”..CustomerService” autowire=”byType”/>
```

Q36) What is wiring? How many ways are available?
Ans:

Q37) What is Auto Wiring? How to implement Auto Wiring?
Ans:

Consider the hello class with 4 constructors as follows in the same order

```
Hello()
Hello(A)
Hello(B)
Hello(A.B)
<bean id=”h” class=”..Hello”autowire=”constructor”/>
```

Q38) Which Constructor will be called with the following Availability of the beans?

A type- 0 beans

B type- 0 beans

Ans: Hello()

Q39) Which constructor will be called with the following Availability of the beans?

A type- 1 beans
B type- 0 beans

Ans: Hello(A)

Q40) Which constructor will be called with the following Availability of the beans?

A type- 0 beans
B type- 1 beans

Ans: Hello(B)

Q41) Which constructor will be called with the following Availability of the beans?

A type- 1 beans
B type- 1 beans

Ans: Hello(A,B)

Q42) Which constructor will be called with the following Availability of the beans?

A type- 2 beans
B type- 1 beans

Ans: Hello(A,B) if A bean is resolved with by Name.
Hello(B) if A bean is not resolved with by Name.

Q43) Which constructor will be called with the following Availability of the beans?

A type- 1 beans
B type- 2 beans

Ans: Hello(A,B) if B bean is resolved with by Name.
Hello(B) if B bean is not resolved with by Name.

Q44) Which Constructor will be called called with the following Availability of the beans?

A type- 2 beans
B type- 2 Beans

Ans: Hello(A,B)
Hello(A,B) if A and B beans are resolved with by Name.
Hello(A) if A bean is resolved with by name.
Hello(B) if B bean is resolved with by name.
Hello() if A,B beans are not resolved with by Name

Q45) Consider the Hello class with 3 constructors as follows in the same order.

Hello()
Hello(A)
Hello(B)

Which Constructor will be called with the following Availability of the beans?

A type- 1 bean
B type- 1 bean
Ans: hello(A) present first in class.

Q46) consider the Hello class with 3 constructors as follows in the same order.
Hello()
Hello(A)
Hello(B)
Which Constructor will be called with the following Availability of the beans?
A type- 1 bean
B type- 1 bean
Ans: hello (B) present first in class.

Q47) consider the Hello class with 3 constructors as follows in the same order.
Hello()
Hello(A,B)
Hello(B,A)
Which Constructor will be called with the following Availability of the beans?
A type- 1 bean
B type- 1 bean
Ans: hello (A,B) present first in class.

Q48) consider the Hello class with 3 constructors as follows in the same order.
Hello()
Hello(B,A)
Hello(A,B)

Which Constructor will be called with the following Availability of the beans?
A type- 1 bean
B type- 1 bean
Ans: hello (B,A)present first in class.

Q49) What will happen with the following code?

```
Class A{ }
Class B extends A{}
Class Hello{
….
}
        <bean id=”ao” class=”…A”/>
        <bean id=”bo” class=”…B”/>
        <bean id=”hello” class=”…Hello” autowire=”by Type”/>
```

Ans: It throws Unsatisfied Dependency Exception. Two Beans found with the A type one is ao and second is bo.
Q50) How can I implement XML Besed Auto wiring?
Ans: Using autowire attribute of bean tag.

Q51) what will happen when I specify the byname autowire process?
Ans:

Q52) what will happen when I specify the by Type autowire process?
Ans:

Q53) what will happen when I specify the constructor autowire process?
Ans:

Q54) what will happen with the following code?

```
        Public class Hai {
                Int a;
                String str;
                Public Hai(int a) {
                        This.a=a;
                }

                Public Hai(String str) {
                        This.str=str;
                }
}
                <beans>
                <bean id=”hai” class=”com.jtc.Hai”>
                <constructor arg value=”1234”/>
                </bean>
                </beans>
```

Ans: Creates the Hai class instance by using Matching Constructor [Hai(String)]

Q55) what will happen with the following code?

```
        Public class Hai {
                Int a;
                String str;
                Public Hai(int a) {
                        This.a=a;
                }
                }
                <beans>
                <bean id=”hai” class=”com.jtc.Hai”>
                <constructor arg value=”1234”/>
                </bean>
                </beans>
```

Ans: Creates the Hai class instance by using Matching Constructor [Hai(int)]

Q56) what will happen with the following code?

```
        Public class Hai {
                Int a;
                String str;
                Public Hai(long str) {
```

```
                System.out.println("Hai-long arg");
                This.str=str;
        }
        Public Hai(int a) {
                System.out.println("Hai-int arg");
                This.a=a;
        }
        }
                <beans>
                <bean id="hai" class="com.jtc.Hai">
                <constructor arg value="1234"/>
                </bean>
                </beans>
```

Ans: Creates the Hai class instance by using constructor which is found first in the class [Hai (long)]

Q57) what will happen with the following code?

```
Public class Hai {
        Int a;
        String str;

        Public Hai (int a) {
                System.out.println("Hai- int arg");
                This.a=a;
        }
        Public Hai(long str) {
                System.out.println("Hai- long arg");
                This. str = str;
        }
        }
                <beans>
                <bean id="hai" class="com.jtc.Hai">
                <constructor arg value="1234"/>
                </bean>
                </beans>
```

Ans: Creates the Hai class instance by using constructor which is found first in the class [Hai (int)]

Q57) what will happen with the following code?

```
Public class Hai {
        Int a;
        String str;

        Public Hai (int a) {
                System.out.println("Hai- int arg");
                This.a=a;
        }
        Public Hai(long str) {
                System.out.println("Hai- long arg");
                This. str = str;
```

```
}
}
        <beans>
        <bean id="hai" class="com.jtc.Hai">
        <constructor arg value="1234"/>
        </bean>
        </beans>
```

Ans: Unsatisfied dependency expressed through constructor argument with index 0 of type [int]: Couly not convert constructor argument value of type [java.lang.String] to required to [int]: Failed to convert value of type java.lang.String' to required type int; nested exception java.lang.Number FormatException: For input string: "jtc"

Q59) How to use log4j in spring application?
Ans: place log4j. properties into src folder.

```
Log4j.properties
        Log4j.rootLogger=DEBUG,jtc
        Log4j.appender.jtc=org.apache.log4j.FileAppender
        Log4j.appender.jtc.file=jtc.log
        Log4j.appender.jtc.layout=org.apache.log4j.patternLayout
        Log4j.appender.jtc.layout.conversionPattern=%p - %m %n
```

Q60 can I use cyclic Dependency injection?
Ans: Depending on the case>

When you are injecting the resources with setter injection then circular reference is allowed. (Refer Q61)

When you are injecting the resources with constructor injection then circular reference may be Allowed or may not (Refer Q62)

Q61) What will happen with the following code?

```
Class hai{
        Hello hello;
        Public void setHello(hello hello) {
                This.hello=Hello;
        }
        Public void show(){
                System.out.println(hello);
        }
}
Public class Hello{
        Hai hai;

        Public void setHai (hai hai){
                This.hai=hai;
}
Public void show(){
        System.out.println(hai);
}
```

```
<beans>
        <bean id="hai" class="com.jtcindia.spring.Hai" autowire=" byType"/>
        <bean id="hello" class=" com.jtcindia.spring.Hello" autowire="byType"/>
</beans>
```

Ans: It will run successfully.
Injects Hai into Hello first and then injects Hello into hai

Q62) What will happen with the following code?

```
        Public class Hai{
                Static{
                        System.out.println("Hai-s.b.");
                }
                Hello hello;
                Public hai(){
                        System.out.println("Hai-D.C");
                }
                Public Hai(Hello hell(Hello hello) {
                        System.out.println("Hai-1 arg");
                        This.hello=hello;
                }
                Public void show(){
                System.out.println (hello);
        }
}
Public class Hello{
        Static{
                System.out.println("Hello-S.B");
        }
        Hai hai;
        Public Hello(){
                System.out.println("hello-D.C");
        }
        Public hello(Hai hai) {
                System.out.println ("Hello-1 arg");
                This.hai=hai;
        }
        Public void show(0{
                System.out.println(hai);
        }
}
        <beans>
        <bean id="hai" class="com.jtcindia. spring.Hai" autowire="constructor"/>
        <bean id="hello" class="com.jtcindia.spring.hello" autowire="constructor"/>
        </beans>
```

Ans: it will run successfully
Only Injects Hello into Hai.

Q63) what is the reason for the following error?
Bean Currently in Creation Exception: Error creating bean with name 'hai' Requested bean is currently in creation: is there an unresolvable circular reference?

Ans: There is some unresolvable circular reference.

Using P-Namespace (property Namespace)

- P-Namespace is designed to reduce the size of spring configuration Document.

```
Class Hell{
        ….
}
classA{
        int a;
        String str;
        Hello hello;
}
```

Without p-Namespace:

```
<bean id="hello" class="…Hello/>
<bean id="ao" class="….A">
        <property name="a" value="111"/>
        <property name="str" value="I am A"/>
        <property name="hello" ref="hello"/>
</bean>
```

With P-Namespace:

```
<bean id="ao" class="….A" p:a="111" p:str="I am A" p:hello-ref="hello"/>
```

Note: you must enable P-Namespace by adding the following in spring configuration document.

```
Xmlns:p=http://www.springframework.org/schema/p
```

Note: in xmlns:p

P is namespace prefix which can be changed.

For example

```
Xmlns:jtc =http://www.springframework.org/schema/p
<bean id="ao" class="…A" jtc:a="111"jtc: str="I am A" jtc:hello-ref="hello"/>
```

Jtc12: Files required

Jtc12.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtc12.java

```
Package com.jtcindia.spring;
Import org.springframework.context.ApplicationContext;
Import org.springframework.context.support.classPathXmlApplicationContext;
Public class Jtc12{
```

```
Public static void main (String[] args){
ApplicationContext ctx=new classpathXmlApplication Context("jtcindia.xml");
System.out.println("…Spring container is now Ready…");
Hello hello=(Hello) ctx.getBean ("hello");
Hello.show(); }  }
```

A.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class A {
Private int a;
Private string msg;
Public void seta(int a) {
This.a=a;
}
Public void setMsg(String msg) {
This.msg=msg;
}
Public void show A (){
System.out.println (a);
System.out.println(msg);
}
}
```

B.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class B {
Private int b;
Private string msg;
Public void seta(int b) {
This.b=b;
}
Public void setMsg(String msg) {
This.msg=msg;
}
Public void show B (){
System.out.println (b);
System.out.println(msg);
}
}
```

Hello.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Hello {
Private A aobj:
Private B bobj:

Public void setAobj(A aobj) {
This.aobj=aobj;
}
Public void setBobj(B aobj) {
This.Bobj=Bobj;
}
Public void show(){
Aobj.showA();
Bobj.showB();
```

Jtcindia.xml

```
<? Xml version="1.0" encoding="UTF-8"?>
<beans
xmls=http://www.springframework.org/schema/beans
Xmlns:jtc=http://www.springframework.org/schema/p
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"jtc:a="10"
Jtc:msg="AAA"/>
<bean id="bobj" class="com.jtcindia.spring.B"
jtc.b="20"
Jtc:str="BBB"/>
```

} }	<bean id=”hello” class=”com.jtcindia.spring.Hello” Jtc:aobj-ref=”aobj” jtc:bobj-ref=”bobj”/> </beans>

Annotation based auto wiring

- When you want to use the Annotations, you need to do the following.
  You must enable context namespace
  Add <context:annotation-config/>
- New you can use @Autowired annotation for the beans.

Ex1:

```
Class Hai{}
Class A{}
Class Hello{
        @Autowired
        A aobj;

        @Autowired
        Hai hai;
…
}

        <bean id=”ha” class=”…Hai”/>
        <bean id=”ao” class=”…A”/>
        <bean id=”hello” class=”…Hello”/>
```

Ex 2:

```
Interface CustomerDAO{}
Class JDBC Customer DAO imp Customer DAO {}
Class Customer Service{
  @Autowired
   Customer DAO cdao = null;
…
}
        <bean id=”jcdao” class=”…JDBCCustomerDAO”/>
        <bean id=”cs” class=”…Customer Service”/>
```

- When you use @Autowired, then by default, beans will be detected based on by Type process and inject them.
- By default, its functionality is same as autowire=”byType”
- When you want to detect the beans based on by Name process then you need to @Qualifier Annotation along with @Autowired.
- When you use autowire=”by Type” then beans will be detected based on Data Type and injects Using setter methods where as when you use @ Autowired then beans will be detected based on Data Type and injects without setter methods.
- When you use autowire=”byname” then beans will be detected based on bean name and injects using setter methods where as when you use @Autowired & @ Qualifier then beans will be detected based on bean and injects without setter methods.

Class Hai {}
Class Hello{
 	
@Autowired
@Qualifier("ha1")
Hai hai;
….
}
	<bean id="ha1" class="….Hai"/>
	
	<bean id="ha2" class="….Hai"/>
	
	<bean id="hello" class="…..hello"/>

Jtc13: Files required

Jtc13.java	A.java
B.java	Hello.java
Jtcindia.xml	

Jtc13.java

```
Package com.jtcindia.spring;
Import org.springframework.context.ApplicationContext;
Import org.springframework.context.support.classPathXmlApplicationContext;
Public class Jtc13{
Public static void main (String[] args){
ApplicationContext ctx=new classpathXmlApplication Context("jtcindia.xml");
Hello hello=(Hello) ctx.getBean ("hello");
Hello.show(); } }
```

A.java

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

Public void seta(int a) {
This.a=a;
}
Public void setMsg(String msg) {
This.msg=msg;
}
Public String to String () {
Return " "+a+"\t"+msg;
}
}
```

B.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class B {
Private int b;
Private string str;
Public void seta(int b, String str) {
This.b=b;
This. Str= str;
}
Public String to String () {
Return " "+b+"\t"+str;
}
}
```

Hello. java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.annotation.Autowired;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Hello {
@Autowired
Private A aobj;

@Autowired
Private B bobj;

Public void show(){
System.out.println(aobj);
System.out.println(bobj);
}
```

Jtcindia.xml

```
<?xml version=”1.0” encoding=”UTF-8”?>
<beans xmlns=http://www.springframework.org/schema/beans
Xmlns:p=http://www.springframework.org/schema/p
Xmlns:context=http://www.springframework.org/schema/context
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
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd”>

<context: annotation-config/>

<bean id=”ao” class=”com.jtcindia.spring.A” p:a=”10” p:msg=”AAA”/>

<bean id=”bo” class=”com.jtcindia.spring.B”>
<constructor-arg value=”20”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”hello” class=”com.jtcindia.spring.Hello”/>
</beans>
```

Jtc14: Files required

Jtc14.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello. java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.annotation.Autowired;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Hello {
Private A aobj;
Private B bobj;

@Autowired
Public Hello(A ao, B bo) {
This.aobj=ao;
This.bobj=bo;
}

Public void show() {
System.out.println(aobj);
System.out.println(bobj);
}
}
```

Jtc15: Files required

Jtc15.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello.java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.annotation.Autowired;
Import org.springframework.beans.factory.annotation.Qualifier;
Public class Hello {
@Autowired
@Qualifier("ao1")
Private A aobj;

@Autowired
@Qualifier("bo1")
Private B bobj;

Public void show() {
System.out.println (aobj);
System.out.println (aobj);
}
}
```

Jtcindia.xml

```
<beans…>
<context:annotation-config/>
<bean id=”ao1” class=”com.jtcindia.spring.A” p:a=”10” p:msg=”A10”/>
<bean id=”ao2” class=”com.jtcindia.spring.A” p:a=”20” p:msg=”A20”/>
<bean id=”bo1” class=”com.jtcindia.spring.B”>
<constructor-arg value=”99”/>
<constructor-arg value=”B99”/>
</bean>
<bean id=”bo2” class=”com.jtcindia.spring.B”>
<constructor-arg value=”88”/>
<constructor-arg value=”B88”/>
</bean >
<bean>
<bean id=”hello”class=”com.jtcindia.spring.hello”/>
</beans>
```

Jtc16: Files required

Jtc16.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello. java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.annotation.Autowired;
Import org.springframework.beans.factory.annotation.Qualifier;

/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Hello {
Private A aobj;
Private B bobj;

@Autowired
Public Hello(@Qualifier(“ao1”) A ao, @qualifier(“bo1”) B bo){
This.aobj=ao;
This.bobj=bo;
}
Public void show() {
System.out.println(aobj);
System.out.println(bobj);
}
}
```

Jtc17: Files required

Jtc17.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello. java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.annotation.Autowired;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Hello {
@Autowired(required=false)
Private A aobj;

@Autowired(required=false)
Private B bobj;

Public void show(){
System.out.println(aobj);
System.out.println(bobj);
}
}
```

Jtcindia.xml

```
<?xml version=”1.0”encoding=”UTF-8”?>
<beans…>
<context:annotation-config/>
<bean id=”hello” class=”com.jtcindia.spring.Hello”/>
</beans>
```

Jtc18: Files required

Jtc18.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello. java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.annotation.Required;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
```

```
*@ visit          : www.jtcindia.org
**/
Public class Hello {

Private A aobj;
Private B bobj;

@Required
Public void setAobj(A aobj) {
System.out.println("Hello-setAobj()");
This.aobj=aobj;
}
@Required
Public void setBobj(B aobj) {
System.out.println("Hello-setBobj()");
This.bobj=bobj;
}
Public void show() {
System.out.println(aobj);
System.out.println(bobj);
}
}
```

Jtcindia.xml

```
<?xml version="1.0" encoding="UTF-8?>
<beans…>
<bean id="ao" class="com.jtcindia.spring.A">
<property name="a" value="10"/>
<property name="msg" value="AAA"/>
</bean>
<bean id="bo" class="com.jtcindia.spring.B">
<constructor-arg value="20"/>
< constructor-arg value="BBB"/>
</bean>
<bean id="hello" class="com.jtcindia.spring.Hello" autowire="by Name"/>
</beans>
```

Using JSR-250 Annotations

- 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 javaee.jar file to project but path.

@postConstruct:

- You can mark any method with @PreDestroy Annotation to call that method before destroying the instance.
- Method which is marked with @PreDestroy Contains the code for cleaning the resources initialized with bean instance.

preDestroy:

- You can mark any method with @PreDestroy Annotation to call that method before destroying the instance.
- Method which is marked with @PreDestroy Contains the code for cleaning the resources initialized with bean instance.

@Resource:

- When you use @ Resource, then beans will be detected either based on by Name or by Type process and injects them.

  When name ettribute is specified for @Resource then uses byname process
  When name attribute is not spectified for @Resource then uses by Type proce

Ex1:

```
Class Hai { }
Class A{}
Class Hello{
@Resource
A aobj;

@Resource (name="ha")
Hai hai;
….
}
        <bean id="ha" class="….Hai"/>
        <bean id="ao" class="….A"/>
        <bean id="hello" class="….hello"/>
```

Ex 2:

```
interface Customer DAO{}
class JDCBCCustomerDAO imp Customer DAO{}
class Hibernate Customer DAO imp Customer DAO{}

class Customer Service{
@Resource(name="hcdao")
customerDao cdao=null;
….
}
        <bean id="jcdao" class="…JDBCCustomer DAO"/>
        <bean id="hcdao" class="…Hibernate Customer DAO"/>
        <bean id="cs" class="…customer Service"/>
```

Jtc18: Files required

Jtc18.java	A.java
B.java	Hello.java
Jtcindia.xml	

Hello. java

```
Package com.jtcindia.spring;
Import javax.annotation.Required;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Hello {
@Resource
Private A aobj;

@resource(name=”bo”)
Private B bobj;

Public vaid show(){
System.out.println(aobj);
System.out.println(bobj);
}
}
```

Q64) what is P-NameSpace?
Ans:

Q65) How can I implement Annotation Based Auto wiring?
Ans: Using @Autowired and @Resource.

Q66) what is difference between @Autowired and @Resource?
Ans:

Q67) can I use Annotations directly in spring application?
Ans: No,
You must enable annotation processing by writing the <context:annotation-config/> tag in spring configuration document.

Q68) How to decide to the type of Dependency Injection?
Ans:

Setter Injection or Filed Injection >Make the beans loosely coupled
Constructor Injection >make the beans tightly coupled

Ex:

```
Class A{}
Class Hai{
A ao;
….
```

```
Hai() {}
Public void setAo (A ao) {
This.ao=ao;
}
}
```

Without A bean instance;
Container will create Hai bean instance
Container cannot inject A bean into hai.
i.e. Without A bean, Hai bean is instantiated. So these two beans are tightly coupled.

Q69) How can I resolve compile time polymorphism with constructor injection?
Ans: Depending on the Availability of the resources and constructors. Refer the Notes and Jtc 5,6,7,

Q70) can I use P-Namespace with constructors?
Ans: No

Q71) What is the bean name?
Ans: bean id is also called as bean name.
You can also specify the name for the bean in the bean definition.

Ex:

```
<bean id="ao" class="com.jtc.spring.A"/>
        Ctx.getbean ("ao");
<bean name="bo" class="com.jtc.spring.B"/>
        Ctx.getBean ("bo");
<bean id="co" name="cobj" class=" com.jtc.spring.c"/>
        Ctx.getBean("co");
        Ctx.getbean("cobj");
```

Spring Container Callbacks or Lifecycle methods

1. Initialization callbacks
2. Disposable callbacks
3. Knowing who you are and where you are.
4. Extending spring container Functionality.

Initialization callbacks:

- You can initialize the Resources required for your bean by using wiring process.

- Sometimes, you may get the requirement to initialize the Resources equired for your bean explicitly or you may want to verify the resources injected by the spring container.
- In this case, you have to write the Resources Initialization code or verification code inside the initialization callbacks.

Spring supports 3 ways for initialization.

1. Write your own initialization method and mark that with @ post Construct annotation.
2. Implement Initalizing Bean interface and override the following method.
   Public void after Properties Set()
3. Write your own initialization method and specify the method name in bean definition (in xml)

Ex:

```
Class hello implements initializing bean{
….
@PostConstruct
Public void init(){
        //initialization code
}
Public void after Properties Set (){
        //initialization code
}
Public void my Init(){
        //initialization code
}
}
```

In XML

```
<bean id=”hello” class=”com.jtc.Hello” init-method=”myInit”/>
```

Disposable callbacks:

- You may get the requirement to clean up the Resources which are initialized at the time of creating the bean instance.
- You write the Resources clean up code inside the Disposable Callbacks.

Spring supports 3 ways for cleanup the Resources.

1. Write your own clean up method and mark that with @preDestroy annotation.
2. Implement Desposable Bean interface and override the following method.
   public void destroy()
3. Write your own clean own clean up method and specify the method name in bean definition.

Ex:

```
Class hello implements DisposableBean {
…
@PreDestroy
Public void cleanup() {
        //cleanup code
}
```

```
Public void destroy() {
        //cleanup code
Public void myCleanup(){
        //cleanup code
}
}
```

In xml

<bean name="hello" class="com.jtc.Hello"destroy-methode="mycleanup"/>

Knowing who you are and where you are

- Sometimes Bean wants to know that its name is and where it is running.
- For this, you can use the following 3 aware interfaces.
    1. Bean Name Aware
    2. Bean Fatory Aware
    3. Application Context Aware
- When Bean wants to know its Name then Bean class has to implement Bean name Aware interface and has to override the following method.
    Public void set Bean Name(String beame)

- When bean wants to get the Bean Factory reference then Bean class has to implement Bean Fatory Aware interface and has to override the following method.
    Public void set Bean Factory (Bean Factory factory)

- When Bean wants to get the Application Context reference then Bean class has to implement Application Context Aware interface and has to override the following method.
    Public void set Application Cation Context (Application Context ctx)

Note:

- Bean Factory and Application Context objects can be injected into the bean in two ways.
    Using Aware interfaces.
    Using @ Autowired

Extending Container Functionality

- You can extend the spring Container Functionality using Bean Post Processor interface.

Steps:

- Write your own Custorm class by implementing Bean post Processor interface.
- Override the following 2 methods.
  Public object po9st process BeforeInitialization (Object bean, string bn)
  Public object post process AfterInitialization (object bean, String bn)

- Register your Bean Post Processor with the spring Container.

Spring Containers

- There are two types to containers provided.
    Bean Factory Container
    Application Context Container

Bean Factory container

- You can create the Bean Factory container as follows.
    - Resource res=new Class Path Resource ("jtcindia. Xml");
    - Resource res=new File System Resource ("D:/D1/Spring/Jtcs/IOC/Jtc21/src/jtcindia.xml")
    - Bean Factory factory = new xml Bean Factory (res);

Life of Bean in the bean Factory container

1. Container loads the bean class into memory.
2. Container creates the Bean instance by using corresponding constructor (constructor injection)
3. Bean dependencies will be injected with the following ways
4. When bean class is implementing Bean name aware interface then set Bean Name() method will be called by the container.
5. When bean class is implementing bean Factory aware interface then set bean factory (0 method will be called byt the container.
6. When bean class is implementing Initializing Bean interface then after Properties Set(0 method will be called by the conteainer.
7. When bean definition contains init method attribute then that specified method will ne called.
8. Fully initialized Bean instance will be ready to use in the Bean Factory cointainedr.
    - At container Shutdown time, it will destroy all the Bean instances.
    - When container is destroying one bean instrance, it will do the following tasks.
1. When bean class is implementing DisposableBean interface the Destroy () method will be called by the container.
2. When bean definition contains destroy-method attribute the that specified method will be called.

Application context container

- Application Context interface has three concrete implementations.
    1. Class path xml application context
    2. File System xml application context
    3. Xml wab application context
- You can create the application context container as follows.
    1. Application Context ctx=new class Path xml Application Context ("jtcindia.xml");
    2. Application Context ctx=new File system xml Application context ("D:/D1/spring/Jtcs/Ioc/lob20/src/jtcindia.xml");

Life of Bean in the Application Context container

1. Container loads the Bean class into memory.
2. Container creates the Bean instance by using corresponding constructor (Constructor injection)
3. Bean Dependencies will be injected with the following ways
   a. Annotation based auto wiring. (Filed Injection)
   b. XML besed Explicit Wiring (Setter Injection)
   c. XML based Auto wiring (Setter Injection)
4. When bean class is implementing Bean Name Aware interface then set Bean Name () method will be called by the container.
5. When bean class is implementing Bean Factory Aware interface then set bean Factory () method will be called by the container.
6. When bean class is implementing Application Context Aware interface then set Application Context () method will be called by the conteainer.
7. When bean po9st processor is registered then post process BeforeInitialization () method will be called by the container.

8. When any method is found with @ post Construct annotation then that method will be called.
9. When be class is implementing Initializing Bean interface then after Properties Set() method will be called by the co9ntainer.
10. What bean definition contains init method attribute then that specified method wil be called.
11. When Bean Post Processor is registered then post process After Initialization () method will be called by the container.
12. Fully initialized bean instance will be ready to use in the Application Context container

1. When any method is found with @ preDestroy annotation then that method will be called.
2. When bean class is implementing Disposable Bean interface then destroy () method will be called by the container.
3. When bean definition contains destroy method attribute then that specified method will be called.

Bean Factory	Application Context
Bean Factory is an interface which has one concrete sub class called Xml Bean Factory.	Application Context is an interface which has 3 concrete sub class called Class Path Xml Application Context File system xml Application Context Xml web application context
Beans configured with Bean Factory container will be loaded lazily by default.	Beans configured with application Context container will be loaded aggressively by default.
Bean factory container does not support annotations.	Application context container supports annotations.
Bean factory container does not support Bean poan post processor	Application context container supports Bean post processor.
Bean factory conteainer does not support Event publishing.	Application context container supports Event publishing.
Bean factory container does not provide the way to resolve message bundles.	Application Context container provides to way to resolve message buldles.

Similarities between Bean Factory container and application Context container

Both the containers will do the following common tasks:

1. Loads bean classes.
2. Create the bean instances.
3. Injects the bean Dependencies.
4. Maintains the bean instances as long as container is running
5. Destroys the Bean Instances at shut down time.

Bean definition

<bean id=””
Name=””
Class=””
Lazy-init=””
scope=””
autowire=””
init-method=””
destroy-method=””

abstarct=””
parcnt=””
…./>

Jtc20: Files required

Jtc20.java	A.java
B.java	Hello.java
Jtcindia.xml	MyBeanPostProcessor.java

Jtc20. java

```
Package com.jtcindia.spring;
Import org.springframework.context.support.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc20 {
Public static void main (String [] args) {
Abstract Application Context ctx = new class Path xml Application context (“jtcindia.xml”);
/*
Abstract Application Context ctx=new
File System xml Application Context (“D:/D1/Spring/Jtcs/IOC/Jtc20/src/jtcindia.xml”):
*/
System.out.println (“spring container is Ready…”);
System .out.println(“……”);
Hello hello=(Hello)ctx.get Bean (“Hello”);
Hello.show();
System.out.println(“……..);
System.out.println(“spring container going to shutdown..”);
Ctx.register Shutdown Hook ();
} }
```

A. java

```
Package com.jtcindia.spring;
Import org.springframework.context.support.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class A {
Private int a;
Private string msg;
Static{
Public A(){
System.out.println(“A-()”);
}
Public void seta(int a) {
This.a=a;
}
Public void set Msg(String msg) {
This.msg=msg;
}
@Post construct
Public void init(){
System.out.println(“A-init()”);
}
```

```
System.opt.println("A-S.B");
}
```

```
Public string to string(){
Return ""+a+"\t"+msg;
}
}
```

B. java

```
Package com.jtcindia.spring;
Import org.springframework.context.support.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class B {
Private int b;
Private String str;
Static{
System.out.println("B-S.B");
}
Public B(){
System.out.println("B-()");
}
Public B(int b, string str){
System.out.println("B-(int, string)");
This.b=b;
This.str=str;
}
@postCopnstruct
Public void init(){
System.out.println("B-init()");
}
Public string to String(){
Return""+b+"\t"+ str:
}
```

My Bean Post Processor.java

```
Package com.jtcindia. spring;
Import org.spring framework. Beans Beans
Exception;
Import org.spring framework. Beans factory
config.Bean Post Prod error;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class My Bean post processor implements
Bean Post processor {
Public object post Process Beforelnitialization
(Object string bname) throw2s bean Exception {
System.out.println("post Process
Beforelnitialization:"+bname);
Return obj;
}
Public object post Process After lnitialization
(object obj, string bname ) throws Bean
Exceptioon{system.out.println("post
processAfterlnitialization:"+beame);
Return obj;
}
}
```

Hello.java

```
Package com.jtcindia.spring;
Import javax.annotation.*;
Import org.springframework.beans.*;
Import org.springframework.beans.factory.*;
Import org.springframework.beans.factory.
annotation .autowir ed;
Import org.springframework.context.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
```

```
@post Construct
Public void init2()");
System.out.println("hello-init2()");
Msg="welcome to jtc ;
If(str==null){
Str="Hai Guys";
}
}

Public void afterPropertiesSet() throws Exception {
System.out.println("Hello-after Properties Set()");
Msg="welcome to jtc"';
If(str==null{
```

```
Public class hello implements Initializingbean, Disposable bean, bean Name Aware,Bean Factory Aware, Application Context Aware{
Str="hai Gays";
}
}
```

```
Int x;
String str;
String msg;
String benam;

@Autowired
A aobj;
B bobj;

Application Context ctx1;
BeanFactory factory1;

@Autowired
Application Context ctx2;
@Autowired
beanFactory factory2;
static{
system.out.println("Hello-S.B");
}
Public hello(int X){
System.out.println("Hello-(int x)");
This.x=x;
}
Public void setStr(string str) {
System.out.println("hello-setStr()");
System.out.println(aobj);
System.out.println(bobj);
This.str=str;
}
Public void set Bobj(B bobj){
System.out.println("hello-setBobj()");
This.bobj=bobj;
}
@post Construct
Public void init1(){
System.out.println("Hello-init1()");
Msg="welcome to jtc"';
If (str==null){
Str="Hai Guys";
}
}
Public void mylnit()[
System.out.println ("Hello-mylnit()");
Msg="welcome to jtc";
If(str==null){
Str="hai Guys"'
}
}
Public void set Bean Name 9string bname) {
System out.println("Hello-setBean Name()");
This.bname
}
Public void set Bean Factory(bean Factory factocn
)
System.out.println(Hello set bean Factory()");
This.factory1=factory1;
}
Public void set Application Context (Application Context ctx1){
System.out.println("hello-set Application Context()");
This.ctx1.=ctx1;
}
@preDestroy
Public void cleanup(){
System.out.println("hello-cleanup()");
}
Public void destroy()throws Exception{
Syste.out.println(Hello-destrohy();
}
public void mycleanup(){
system.out.println("Hello-mycleanup()");
}
Public void show(){
Syste.out.println("Hello-show()");
Syste.out.println(x);
Syste.out.println(str);
Syste.out.println(msg);
Syste.out.println("bean Name is+"bname);
Syste.out.println(factory!);
Syste.out.println(ctx1);
Syste.out.println(cactory2);
Syste.out.println(ctx2);
Syste.out.println(ctx1==ctx2);
```

	Syste.out.println(factory1==factory2); } }

Jtcindia.xml

```
<?xml version=”1.0” encoding=”UTF-8”?>
<beans…>
<context:annotation-config/>
<bean id=”AO” class=”com.jtcindia.spring.A”>
<property name=”a” value=”99”/>
<property name=”msg”value=”AAA”/>
</bean>
<bean id=”bobj” class=”jtcindia.spring.B”>
<constructor-arg value=”88”/>
<constructor-arg value=”BBB”/>
</bean>
<bean id=”HELLO” class=”com.jtcindia.spring.Hello” init-method=”mylnit”
Destroy-method=”mycleanup” autowire=”byname”>
<constructor-arg value=”99”/>
<property name=”str” value=”Hello Guys”/>
</bean>
<bean class=”com.jtcindia.spring.MyBean Post Processor”/>
</beans>
```

Jtc21: Files required

Jtc21.java	A.java
B.java	Hello.java
Jtcindia.xml	MyBeanPostProcessor.java

Jtc21.java

```
Package com.jtcindia.spring;
Import org.springframework.beans.factory.beanFactory;
Import org.springframework.beans.factory.xml.xmlBeanFactory;
Import org.springframework.core.io.*;
Public class Jtc21 {
Public static void main (string[] args) {
Resource res=new Class Path Resource (“jtcindia.xml”);
Bean Factory factory=new xml bean Factory (res);
System.out.println (“spring container is Ready…”);
System .out.println(“……”);
Hello hello=(Hello)factory.get Bean (“Hello”);
Hello.show();
System.out.println(“……..);
System.out.println(“spring container going to shutdown..”);
```

```
} }
```