Spring Class Notes

Spring Practice -First Program:

1) Spring container takes two inputs like Spring Bean and Configuration File. After Reading them, Spring container instantiates and assigns the values.
2) By default it creates singleton Objects in Container.
3) To read the Object use getBean(String beanName):Object method.

Step 1) Create New Java Project and Add build path (Spring JARS and +Commons-logging JAR)

Spring Download Location:

https://repo.spring.io/release/org/springframework/spring/3.2.5.RELEASE/spring-framework-3.2.5.RELEASE-dist.zip

(Extract and Find JARS in dist folder)

Commons-Logging JAR:

http://central.maven.org/maven2/commons-logging/commons-logging/1.1.1/commons-logging-1.1.1.jar

>> Click on File->New -> Java Project.

>> Provide Any Sample Name (ex: SpringCore) click on Finish Button

>> Right Click on Project -> Choose Build Path -> Configure buildPath-> Libraries Tab->Add External JARS (Select above downloaded JARS). Even add Commons-logging jar.

Step 2) Right Click on src folder -> New Class-> Enter package and Class name:

Ex:

```
package com.app;

public class Employee {
	private int empId;
	private String empName;
	private double empSal;
	public Employee() {
		super();
	}
```

```
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" +
empName
				+ ", empSal=" + empSal + "]";
	}
}
```

Step 3) Create A Spring configuration file to configure the Spring Bean Class.
Right Click on src->new->other->type XML-> select XML File-> Next->
Enter any name.xml (ex: config.xml)->Finish

Double click on file to open and goto source tab.
And provide configuration for Employee.java
Ex:

```
<?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/bean
s
	http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

	<bean class="com.app.Employee" name="emp">
		<property name="empId">
			<value>10</value>
		</property>
		<property name="empName">
			<value>ABCD</value>
		</property>
```

```
            <property name="empSal">
                  <value>732.36</value>
            </property>
      </bean>
</beans>
```

Step 4) Create a new Test Class to Access the above Object.

Right Click on src->new->class Enter any name (Ex:Main.java)

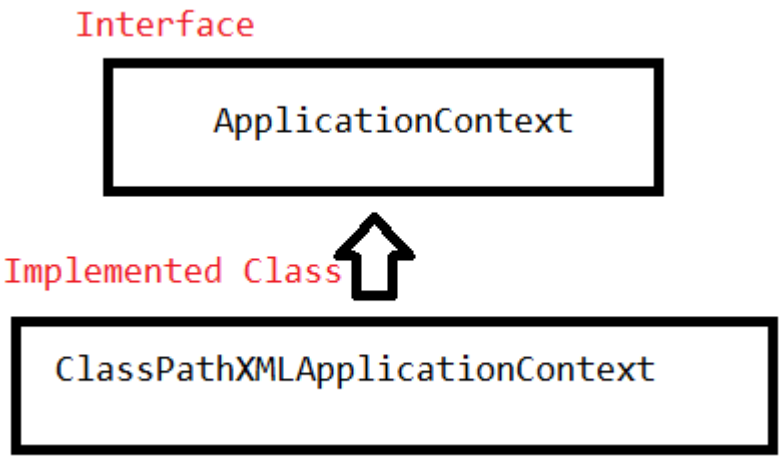
```
package com.app;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.app.module.Employee;


public class Main {


      public static void main(String[] args) throws Exception,
Exception  {
            ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
            Employee emp=(Employee)context.getBean("emp");
            System.out.println(emp);

      }

}
```

Note: Here ApplicationContext is an interface. It represents Spring Container. And It's Implementation class is ClassPathXMLApplicationContext.

>> getBean(String beanName):Object return required Objects in up-casted format. Again we need to downcast them.

Some Core Java Concepts used in the above Coding:
Up-casting: Referring Sub class object using Super Type(Class or interface). To do up casting classes must have inheritance relation.
Down-casting: Already Up casted Object again converting to Sub type is known as Down-casting.

Here



Note: Eclipse must be under Java Perspective.

Eclipse Shortcuts : Alt+Shift+S R (Setters and Getters), Alt+Shift+S S (toString method),Alt+Shift+S O (Constructors).

Spring Introduction Notes 2:

Spring Bean Primitive type setter injection code can be written in 3 ways. They are

i)Value as Tag(<value></value>)

ii)Value as Attribute(<property name="" value=""/>)

iii)p-Name space(<bean name="" class="" p:variable-name="value"/>)

ex:

Spring Bean:

```
packagecom.app.module;

publicclass Employee {
	privateintempId;
	private String empName;
	privatedoubleempSal;
	public Employee() {
		super();
	}
	publicintgetEmpId() {
		returnempId;
	}
```

```
	publicvoidsetEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoidsetEmpName(String empName) {
		this.empName = empName;
	}
	publicdoublegetEmpSal() {
		returnempSal;
	}
	publicvoidsetEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
				+ ", empSal=" + empSal + "]";
	}



}
```

For the Above Spring bean Configuration code can be written as:

i)Value as tag: (Spring Configuration XML Code)

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="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">

	<beanclass="com.app.module.Employee"name="emp">
		<propertyname="empId">
			<value>10</value>
		</property>
		<propertyname="empName">
			<value>ABCD</value>
		</property>
		<propertyname="empSal">
			<value>732.36</value>
		</property>
	</bean>
</beans>
```

ii)Value as attribute:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

      <beanclass="com.app.module.Employee"name="emp">
            <propertyname="empId"value="10"/>
            <propertyname="empName"value="abcd"/>
            <propertyname="empSal"value="23.36"/>
      </bean>
</beans>
```

iii)p-Name Space(p-Schema):

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

      <beanclass="com.app.module.Employee"name="emp"p:empId="101"p:e
mpName="ABCD"p:empSal="20.36"/>


</beans>
```

(Observe the red colour line. That is need to be added while using p-schema)

Note: a) For a dependency , we cannot configure a property more than one time

Ex:

```
<beanclass="com.app.module.Employee"name="emp"p:empId="101">
            <propertyname="empId">
                  <value>101</value>
            </property>
</bean>
```

```
Main Class:

Public class Main{

publicstaticvoid main(String[] args) throws Exception, Exception  {
```

```
            ApplicationContext
context=newClassPathXmlApplicationContext("config.xml");
            Employee emp=(Employee)context.getBean("emp");
            System.out.println(emp);

      }

}
```

Output:

```
xception in thread "main"
org.springframework.beans.factory.parsing.BeanDefinitionParsingExcep
tion: Configuration problem: Property 'empId' is already defined
using both <property> and inline syntax. Only one approach may be
used per property.
```

```
<beanclass="com.app.module.Employee"name="emp">
            <propertyname="empId">
                  <value>101</value>
            </property>
            <propertyname="empId">
                  <value>102</value>
            </property>
</bean>
```

Exception:

```
org.springframework.beans.factory.parsing.BeanDefinitionParsingExcep
tion: Configuration problem: Multiple 'property' definitions for
property 'empId'
```

Note: b) By default <value> tag takes input as String type and later it will be parsed to specific type. If Data can't be parsed (or any mismatch) then conversion problem will occur, and appropriate exception will be thrown.

Like Number format Exception.

Ex: empId is int type and sending data as "ABCD"

```
<beanclass="com.app.module.Employee"name="emp">
            <propertyname="empId">
                  <value>ABCD</value>
            </property>

</bean>
```

```
Exception in thread "main"
org.springframework.beans.factory.BeanCreationException: Error
creating bean with name 'emp' defined in class path resource
```

```
[config.xml]: Initialization of bean failed; nested exception is
org.springframework.beans.TypeMismatchException: Failed to convert
property value of type 'java.lang.String' to required type 'int' for
property 'empId'; nested exception is
java.lang.NumberFormatException: For input string: "ABCD"
```

Note: c) getBean(String beanName,Classclasstype) is a overloaded method, which is used to read a bean from spring container with any down casting.

Ex:

```
publicstaticvoid main(String[] args) throws Exception, Exception  {
            ApplicationContext
context=newClassPathXmlApplicationContext("config.xml");
            Employee
emp=(Employee)context.getBean("emp",Employee.class);
            System.out.println(emp);

}
```

Spring Collections:

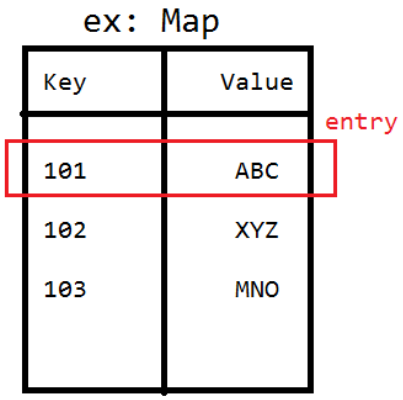
In Spring basically collections are configured as

1) List
2) Set
3) Map
4) Properties

To specify above collections , tag are <list>,<set>,<map> and <props>

Here <map> contains internally entries (<entry>). Every entry contains <key> and <value> , these even can be represents as attributes.

Ex:



Properties also stores data in the key value pair only. But by default key and values are type String in Properties. It contains child tag <prop key=””></prop>. It doesn’t contain any <value> tag to represent the value; we can directly specify the value in between the <prop> tag.

Ex:

Employee.java

```
packagecom.app.module;

importjava.util.List;
importjava.util.Map;
importjava.util.Properties;
importjava.util.Set;

publicclass Employee {
	private List<String>empList;
	private Set<String>empSet;
	private Map<String,String>empMap;
	private Properties empProperties;
	public List<String>getEmpList() {
		returnempList;
	}
	publicvoidsetEmpList(List<String>empList) {
		this.empList = empList;
	}
	public Set<String>getEmpSet() {
		returnempSet;
	}
	publicvoidsetEmpSet(Set<String>empSet) {
		this.empSet = empSet;
	}
	public Map<String, String>getEmpMap() {
		returnempMap;
	}
```

```
	publicvoidsetEmpMap(Map<String, String>empMap) {
		this.empMap = empMap;
	}
	public Properties getEmpProperties() {
		returnempProperties;
	}
	publicvoidsetEmpProperties(Properties empProperties) {
		this.empProperties = empProperties;
	}
	@Override
	public String toString() {
		return"Employee [empList=" + empList + ", empSet=" +
empSet
				+ ", empMap=" + empMap + ", empProperties=" +
empProperties
				+ "]";
	}


}
```

Spring Configuration File :config.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

	<beanclass="com.app.module.Employee"name="emp">
		<propertyname="empList"><!-- it will maintain duplicates
-->
			<list>
				<value>A</value>
				<value>A</value>
				<value>A</value>
				<value>B</value>
				<value>C</value>
				<value>D</value>
			</list>
		</property>
		<propertyname="empSet"><!-- it will not maintain
duplicates -->
			<set>
				<value>A</value>
```

```
                        <value>A</value>
                        <value>A</value>
                        <value>B</value>
                        <value>C</value>
                        <value>D</value>
                  </set>
            </property>
            <propertyname="empMap">
                  <map>
                        <entry>
                              <key>
                                    <value>K1</value>
                              </key>
                              <value>V1</value>
                        </entry>
                        <entrykey="K2"value="V2"/>
                        <entrykey="K3">
                              <value>V3</value>
                        </entry>
                        <entryvalue="V4">
                              <key>
                                    <value>K4</value>
                              </key>
                        </entry>
                  </map>
            </property>
            <propertyname="empProperties">
                  <props>
                        <propkey="key1">val1</prop>
                        <propkey="key1">val3</prop>
                        <propkey="key2">val2</prop>
                  </props>
            </property>
      </bean>


</beans>
```

Main Program:

```
publicclass Main {
publicstaticvoid main(String[] args) throws Exception, Exception  {
            ApplicationContext
context=newClassPathXmlApplicationContext("config.xml");
            Employee emp=(Employee)context.getBean("emp");
            System.out.println(emp);

      }

}
```

Output:

```
Employee [empList=[A, A, A, B, C, D], empSet=[A, B, C, D],
empMap={K1=V1, K2=V2, K3=V3, K4=V4}, empProperties={key2=val2,
key1=val3}]
```

Note 1):
Empty Object Reference and Null Reference:-
If any Object created with no data and referred with a reference is known as empty Object.
For ex: List l1=new ArrayList(); Creating empty Array List.
If a reference not even contains empty object. Or reference, that doesn't point to Object is known as null Reference.

Ex:

```
packagecom.app.module;

importjava.util.List;

publicclass Employee {
	private List<String>empList;
	public List<String>getEmpList() {
		returnempList;
	}
	publicvoidsetEmpList(List<String>empList) {
		this.empList = empList;
	}
	@Override
	public String toString() {
		return"Employee [empList=" + empList + "]";
	}
}
```

For the above class, configuration code is given as:

Case 1)

```
<beanclass="com.app.module.Employee"name="emp">

</bean>
```

Employee Object Created with empList as null

Case2)

```
<beanclass="com.app.module.Employee"name="emp">
		<propertyname="empList">
			<list></list>
		</property>
</bean>
```

Employee Object created with empList with zero size (Or empty List)

Case 3)

```
<beanclass="com.app.module.Employee"name="emp">
        <propertyname="empList"></property>
</bean>
```

In case of any property tag is defined without any value tag, then Spring Container throws Exception:

org.springframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: <property> element for property 'empList' must specify a ref or value

Spring Notes: Reference Type Dependency

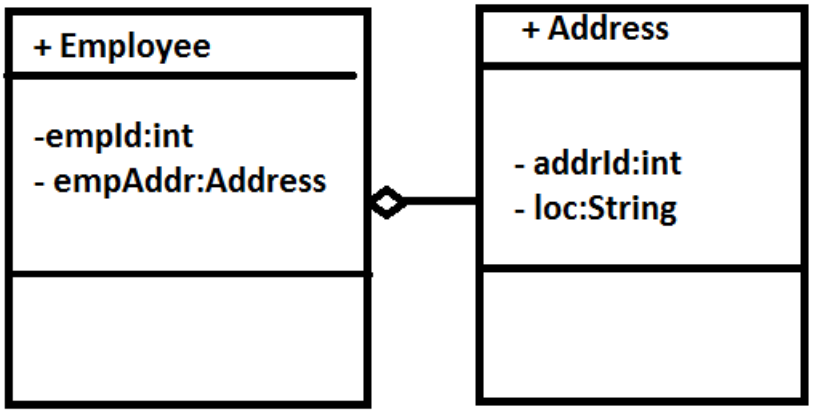
A Class can have relation with another class (HAS-A) dependency. To configure that in Spring Configuration File (config.xml) <ref/> tag is used.

Syntax:

<ref bean="already existed bean name with same type "/>

In this case we must configure always independent class first, then configure dependent class

ex:




```
package com.app;

public class Employee {
```

```
	private int empId;
	private Address empAddr;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public Address getEmpAddr() {
		return empAddr;
	}
	public void setEmpAddr(Address empAddr) {
		this.empAddr = empAddr;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empAddr=" +
empAddr + "]";
	}

}
========

package com.app;

public class Address {

	private int addrId;
	private String loc;
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}

}
```

=====

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:p="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 class="com.app.Address" name="addrObj">
		<property name="addrId" value="101"/>
		<property name="loc" value="abcd"/>
	</bean>
	<bean class="com.app.Employee" name="empObj">
		<property name="empId" value="1010"/>
		<property name="empAddr">
			<ref bean="addrObj"/>
		</property>
	</bean>
</beans>
```

==========

```
package com.app;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

	public static void main(String[] args) {
		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		Employee employee=context.getBean("empObj",
Employee.class);
		System.out.println(employee);
	}
}
```

==

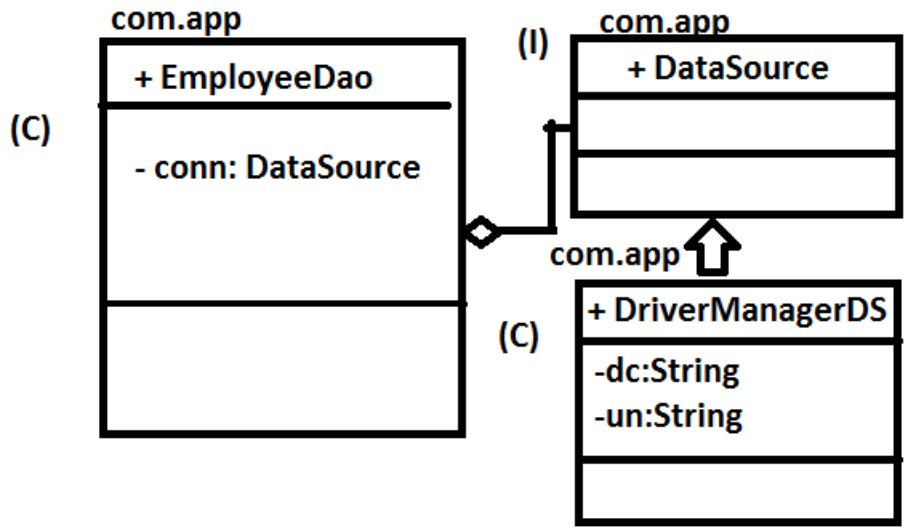
Output:

```
Employee [empId=1010, empAddr=Address [addrId=101, loc=abcd]]
```

==========

Note: If a class is having interface dependency , then class must be configured with implemented class.

ex:



Here DataSource is interface. So we cannot configure that, so, DriverManagerDS can be injected to EmployeeDao.

Now XML Code is:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:p="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 class="com.app.DriverManagerDS" name="dsObj">
		<property name="dc" value="oracleDC"/>
		<property name="un" value="system"/>
	</bean>
	<bean class="com.app.EmployeeDao" name="empDao">
		<property name="conn">
			<ref bean="dsObj"/>
		</property>
	</bean>
</beans>
```

Spring : Reference Type Dependency:

Spring Reference Type Dependency Can be configured as 4 ways:

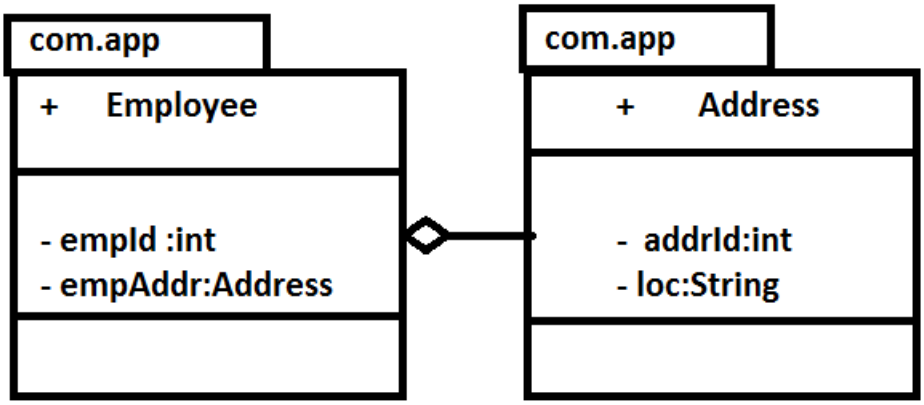
1) ref as tag (<**ref bean**=""/>)

2) ref as Attribute (<property name="" **ref**=""/>)

3) p-schema (<bean name="" class="" **p:variable-ref**="">)

4) inner bean (<bean..><property..><bean..>
</bean></peoperty></bean>)

ex:



Java Code:

Address.java

```
package com.app;

public class Address {

	private int addrId;
	private String loc;
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
```

```
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}


}
```

==========

Employee.java

```
package com.app;

public class Employee {

	private int empId;
	private Address empAddr;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public Address getEmpAddr() {
		return empAddr;
	}
	public void setEmpAddr(Address empAddr) {
		this.empAddr = empAddr;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empAddr=" +
empAddr + "]";
	}


}
```

Spring Configuration File:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

```
	<bean class="com.app.Address" name="addrObj">
		<property name="addrId">
			<value>1010</value>
		</property>
		<property name="loc" value="ABCD"/>
	</bean>

	<bean class="com.app.Employee" name="empObj">
		<property name="empId" value="101"/>
		<property name="empAddr">
			<ref bean="addrObj"/>
		</property>
	</bean>

</beans>
```

Test.java

```
package com.app;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;


public class Test {

	public static void main(String[] args) {
		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		Employee employee=context.getBean("empObj",
Employee.class);
		System.out.println(employee);
	}
}
```

output: Employee [empId=101, empAddr=Address [addrId=1010, loc=ABCD]]

4 Ways Of Spring Configuration code for reference type is given below :

1) ref as tag:

```
<bean class="com.app.Employee" name="empObj">
		<property name="empId" value="101"/>
		<property name="empAddr">
			<ref bean="addrObj"/>
		</property>
	</bean>
```

2) ref as attribute:

```
<bean class="com.app.Employee" name="empObj">
      <property name="empId" value="101"/>
      <property name="empAddr" ref="addrObj"/>
</bean>
```

3) p-schema:

```
<bean class="com.app.Employee" name="empObj" p:empAddr-
ref="addrObj">
      <property name="empId" value="101"/>
</bean>
```

```
Note: <beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

p schema line must be added in XML File.

4) Inner Bean:

```
<bean class="com.app.Employee" name="empObj">
      <property name="empId" value="101" />
      <property name="empAddr">
            <bean class="com.app.Address" name="addrObj">
                  <property name="addrId">
                        <value>1010</value>
                  </property>
                  <property name="loc" value="ABCD" />
            </bean>
      </property>
</bean>
```

==========

equals() Method:

equals is a non-final,no-static,non-private(public method) method defined in java.lang.Object class.

it compares two objects memory location. It returns true if two references refers same object, but not similar Objects. Two make equals comparing data (as similar Objects), just override that method and write logic of Objects comparison.

For ex:

At the time of adding References Objects to Set data, which stores Objects of a class that doesn't override the equals method, then if we add similar objects, even it treats as different object.

(Always remember equals method by default check either same or not, that means Two Object must refer same Object. To make equals comparing similar or not, override the method)

Ex: Without equals:

Program and Output:

Address.java

```
package com.app;

public class Address {

	private int addrId;
	private String loc;
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}


}
```

Test.java

```
package com.app;

import java.util.HashSet;
import java.util.Set;

public class Test {
```

```
	public static void main(String[] args) {
		Address address1=new Address();
		address1.setAddrId(101);
		address1.setLoc("ABCD");

		Address address2=new Address();
		address2.setAddrId(101);
		address2.setLoc("ABCD");

		Address address3=new Address();
		address3.setAddrId(101);
		address3.setLoc("ABCD");


		Set<Address> addrSet=new HashSet<Address>();
		addrSet.add(address1);
		addrSet.add(address2);
		addrSet.add(address3);

		System.out.println(addrSet);
	}
}
```

output: [Address [addrId=101, loc=ABCD], Address [addrId=101, loc=ABCD], Address [addrId=101, loc=ABCD]]

Ex: With equals:

Note : To generate equals and hashCode method use shortcut in eclipse (alt+shift+s and h)

```
package com.app;

public class Address {

	private int addrId;
	private String loc;
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
```

```
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + addrId;
		result = prime * result + ((loc == null) ? 0 :
loc.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Address other = (Address) obj;
		if (addrId != other.addrId)
			return false;
		if (loc == null) {
			if (other.loc != null)
				return false;
		} else if (!loc.equals(other.loc))
			return false;
		return true;
	}


}
```

Test.java:

```
package com.app;

import java.util.HashSet;
import java.util.Set;

public class Test {

	public static void main(String[] args) {
		Address address1=new Address();
```

```
            address1.setAddrId(101);
            address1.setLoc("ABCD");

            Address address2=new Address();
            address2.setAddrId(101);
            address2.setLoc("ABCD");

            Address address3=new Address();
            address3.setAddrId(101);
            address3.setLoc("ABCD");


            Set<Address> addrSet=new HashSet<Address>();
            addrSet.add(address1);
            addrSet.add(address2);
            addrSet.add(address3);

            System.out.println(addrSet);
      }
}
```

output:[Address [addrId=101, loc=ABCD]]

Spring Collections with References:

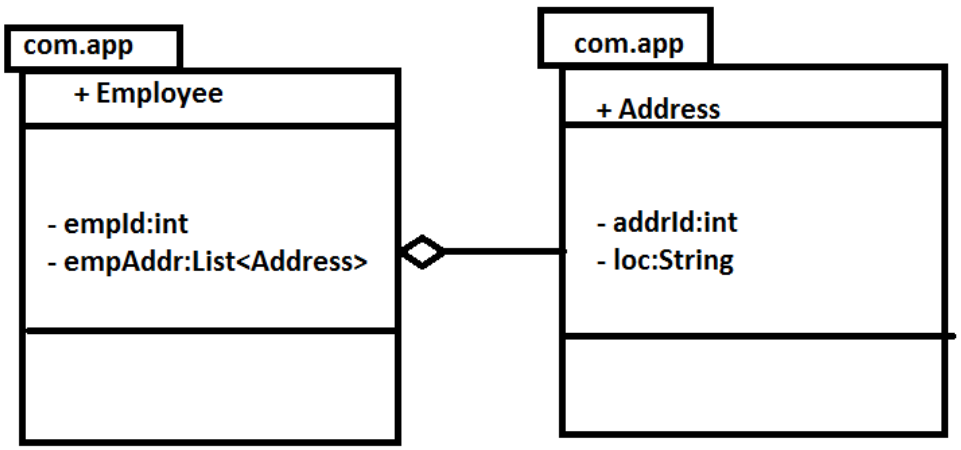
Collection with reference Type can be configured as

1) Inner bean mode

2) External Bean as Reference

3) Stand Alone Collections

ex:



Java Code:

Address.java:

```
package com.app;

public class Address {

	private int addrId;
	private String loc;
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + addrId;
		result = prime * result + ((loc == null) ? 0 :
loc.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Address other = (Address) obj;
		if (addrId != other.addrId)
			return false;
```

```
			if (loc == null) {
				if (other.loc != null)
					return false;
			} else if (!loc.equals(other.loc))
				return false;
			return true;
		}


}
```

Employee.java

```
package com.app;

import java.util.List;

public class Employee {

	private int empId;
	private List<Address> empAddr;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public List<Address> getEmpAddr() {
		return empAddr;
	}
	public void setEmpAddr(List<Address> empAddr) {
		this.empAddr = empAddr;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empAddr=" +
empAddr + "]";
	}


}
```

Configuration code:

1) Inner Bean Mode:-

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
```

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


      <bean class="com.app.Employee" name="empObj">
            <property name="empId" value="101"/>
            <property name="empAddr">
                  <list>
                        <bean class="com.app.Address">
                              <property name="addrId" value="10"/>
                              <property name="loc" value="HYD"/>
                        </bean>
                        <bean class="com.app.Address">
                              <property name="addrId" value="20"/>
                              <property name="loc" value="HYD2"/>
                        </bean>
                        <bean class="com.app.Address">
                              <property name="addrId" value="30"/>
                              <property name="loc" value="HYD1"/>
                        </bean>
                  </list>
            </property>
      </bean>

</beans>
```

2) External Bean as Reference:-

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

      <bean class="com.app.Address" name="addr1">
            <property name="addrId" value="10" />
            <property name="loc" value="HYD" />
      </bean>
      <bean class="com.app.Address" name="addr2">
            <property name="addrId" value="20" />
            <property name="loc" value="HYD2" />
      </bean>
      <bean class="com.app.Address" name="addr3">
            <property name="addrId" value="30" />
```

```
		<property name="loc" value="HYD1" />
	</bean>
	<bean class="com.app.Employee" name="empObj">
		<property name="empId" value="101" />
		<property name="empAddr">
			<list>
				<ref bean="addr1"/>
				<ref bean="addr2"/>
				<ref bean="addr3"/>
			</list>
		</property>
	</bean>

</beans>
```

3) Util Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"
xmlns:util="http://www.springframework.org/schema/util"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

	http://www.springframework.org/schema/util

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

	<util:list id="list1" list-class="java.util.LinkedList">
		<bean class="com.app.Address" name="addr1">
			<property name="addrId" value="10" />
			<property name="loc" value="HYD" />
		</bean>
		<bean class="com.app.Address" name="addr2">
			<property name="addrId" value="20" />
			<property name="loc" value="HYD2" />
		</bean>
		<bean class="com.app.Address" name="addr3">
			<property name="addrId" value="30" />
			<property name="loc" value="HYD1" />
		</bean>
	</util:list>
	<bean class="com.app.Employee" name="empObj">
```

```
            <property name="empId" value="101" />
            <property name="empAddr">
                  <ref bean="list1"/>
            </property>
      </bean>

</beans>
```

Test.java:

```
package com.app;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;


public class Test {

      public static void main(String[] args) {
            ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
            Employee employee=context.getBean("empObj",
Employee.class);
            System.out.println(employee);
            /*to know the implemented class type of list*/

      System.out.println(employee.getEmpAddr().getClass().getName())
;
      }
}
```

output:

Ex:

```
Employee [empId=101, empAddr=[Address [addrId=10, loc=HYD], Address
[addrId=20, loc=HYD2], Address [addrId=30, loc=HYD1]]]
java.util.LinkedList
```

Spring Map Configuration:

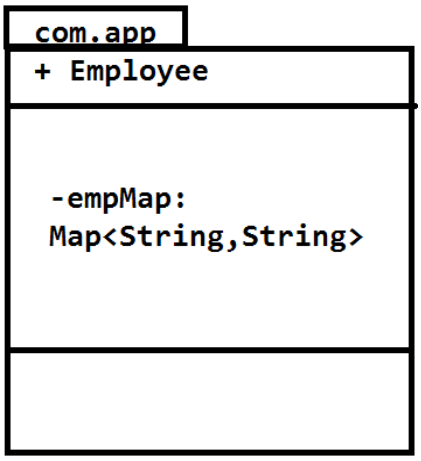
1) Tag based (external bean reference)
2) Inner bean
3) Standalone collections

1) Tag based (external reference)

Map can have key value pair in 4 combinations, and every combination can be configured in 4 ways.

a) Map Key as Primitive and value as Primitive
b) Map Key as Primitive and value as Reference
c) Map Key as Reference and value as Primitive
d) Map Key as Reference and value as Reference

Ex: a) Map Key as Primitive and value as Primitive : Example
Configuration code is given below,



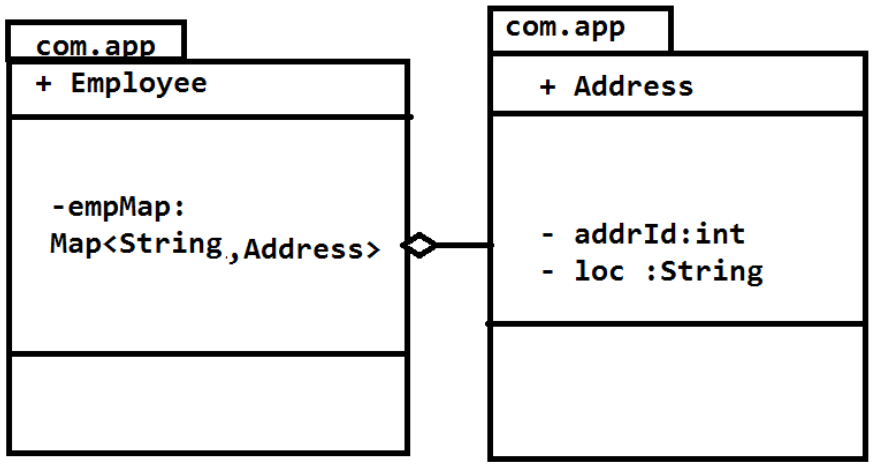
Spring configuration Code:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

      <beanclass="com.app.Employee"name="empObj">
            <propertyname="empMap">
```

```
                <map>
                      <entry>
                            <key>
                                  <value>Key1</value>
                            </key>
                            <value>Val1</value>
                      </entry>
                      <entrykey="Key2">
                            <value>Val2</value>
                      </entry>
                      <entryvalue="Val3">
                            <key>
                                  <value>Key3</value>
                            </key>
                      </entry>
                      <entrykey="Key4"value="Val4"/>
                </map>
          </property>
     </bean>


</beans>
```

b) Map Key as Primitive and value as Reference: Example

Configuration code :

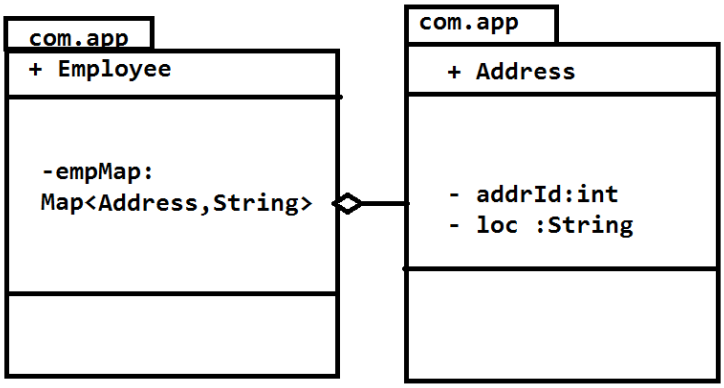



```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">
```

```
<beanclass="com.app.Address"name="addr1">
      <propertyname="addrId"value="110"/>
      <propertyname="loc"value="HYD1"/>
</bean>
<beanclass="com.app.Address"name="addr2">
      <propertyname="addrId"value="120"/>
      <propertyname="loc"value="HYD2"/>
</bean>
<beanclass="com.app.Address"name="addr3">
      <propertyname="addrId"value="130"/>
      <propertyname="loc"value="HYD3"/>
</bean>
<beanclass="com.app.Address"name="addr4">
      <propertyname="addrId"value="140"/>
      <propertyname="loc"value="HYD4"/>
</bean>
<beanclass="com.app.Employee"name="empObj">
      <propertyname="empMap">
            <map>
                  <entry>
                        <key>
                              <value>Key1</value>
                        </key>
                        <ref bean="addr1"/>
                  </entry>
                  <entrykey="Key2">
                        <ref bean="addr2"/>
                  </entry>
                  <entryvalue-ref="addr3">
                        <key>
                              <value>Key3</value>
                        </key>
                  </entry>
                  <entrykey="Key4"value-ref="addr4"/>
            </map>
      </property>
</bean>
</beans>
```

c) Map Key as Reference and value as Primitive: example
Configuration code:

Note: If key is reference type (class) , that class must override hashCode and equals methods(alt+shift+s h)

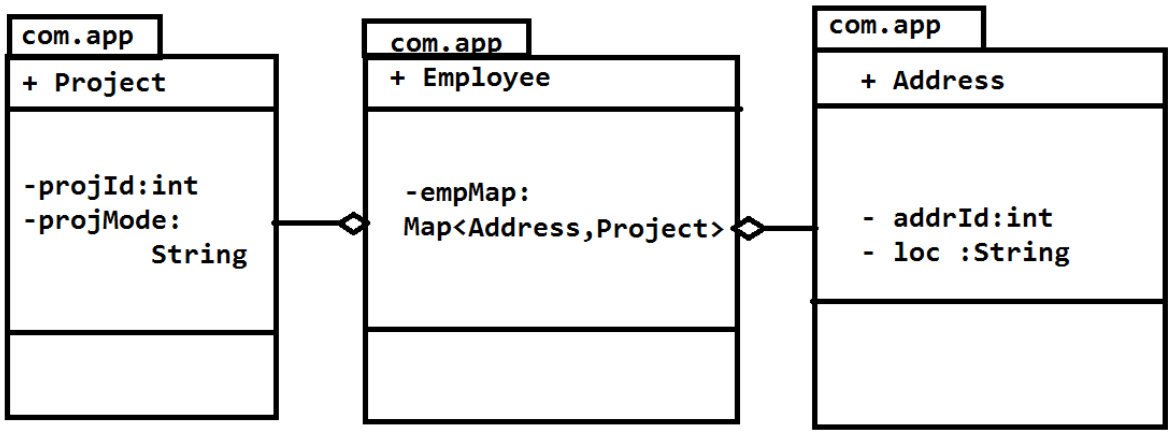



```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

      <beanclass="com.app.Address"name="addr1">
            <propertyname="addrId"value="110"/>
            <propertyname="loc"value="HYD1"/>
      </bean>
      <beanclass="com.app.Address"name="addr2">
            <propertyname="addrId"value="120"/>
            <propertyname="loc"value="HYD2"/>
      </bean>
      <beanclass="com.app.Address"name="addr3">
            <propertyname="addrId"value="130"/>
            <propertyname="loc"value="HYD3"/>
      </bean>
      <beanclass="com.app.Address"name="addr4">
            <propertyname="addrId"value="140"/>
            <propertyname="loc"value="HYD4"/>
      </bean>
      <beanclass="com.app.Employee"name="empObj">
            <propertyname="empMap">
                  <map>
                        <entry>
                              <key>
                                    <ref bean="addr1"/>
                              </key>
                                    <value>Val1</value>
                        </entry>
                        <entrykey-ref="addr2">
                              <value>val2</value>
                        </entry>
                        <entryvalue="val3">
```

```
                        <key>
                            <refbean="addr3"/>
                        </key>
                    </entry>
                    <entry key-ref="addr4"value="val4"/>
                </map>
            </property>
        </bean>
    </beans>
```

d) Map Key as Reference and value as Reference:

Example Configuration Code:




```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

      <beanclass="com.app.Project"name="proj1">
            <propertyname="projId"value="210"/>
            <propertyname="projMode"value="A1"/>
      </bean>
      <beanclass="com.app.Project"name="proj2">
            <propertyname="projId"value="220"/>
            <propertyname="projMode"value="A2"/>
      </bean>
      <beanclass="com.app.Project"name="proj3">
            <propertyname="projId"value="230"/>
            <propertyname="projMode"value="A3"/>
      </bean>
      <beanclass="com.app.Project"name="proj4">
            <propertyname="projId"value="240"/>
            <propertyname="projMode"value="A4"/>
```

```
</bean>
<beanclass="com.app.Address"name="addr1">
      <propertyname="addrId"value="110"/>
      <propertyname="loc"value="HYD1"/>
</bean>
<beanclass="com.app.Address"name="addr2">
      <propertyname="addrId"value="120"/>
      <propertyname="loc"value="HYD2"/>
</bean>
<beanclass="com.app.Address"name="addr3">
      <propertyname="addrId"value="130"/>
      <propertyname="loc"value="HYD3"/>
</bean>
<beanclass="com.app.Address"name="addr4">
      <propertyname="addrId"value="140"/>
      <propertyname="loc"value="HYD4"/>
</bean>
<beanclass="com.app.Employee"name="empObj">
      <propertyname="empMap">
            <map>
                  <entry>
                        <key>
                              <refbean="addr1"/>
                        </key>
                              <refbean="proj1"/>
                  </entry>
                  <entrykey-ref="addr2">
                        <refbean="proj2"/>
                  </entry>
                  <entryvalue-ref="proj3">
                        <key>
                              <refbean="addr3"/>
                        </key>
                  </entry>
                  <entrykey-ref="addr4"value-ref="proj4"/>
                  </map>
      </property>
</bean>
</beans>
```

---

2) Inner Bean:

Instead of referring the bean from key tag or ref tag, we can even declare the bean inside the key or entry.

Ex:

```
<beanclass="com.app.Employee"name="empObj">
            <propertyname="empMap">
<map>
```

```
<entry>
      <key>
            <beanclass="com.app.Project"name="proj1">
                  <propertyname="projId"value="210"/>
                  <propertyname="projMode"value="A1"/>
            </bean>
      </key>
      <beanclass="com.app.Address"name="addr1">
            <propertyname="addrId"value="110"/>
            <propertyname="loc"value="HYD1"/>
      </bean>
</entry>
</map>
</property>
</bean>
```

Here bean declared inside the key represents Key bean and bean declared outside key and inside the entry represent value bean.

---

3) Standalone collections

We can declare a specific collection type using util schema and can be referred from the property tag as a bean.

Ex:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"xmlns:util="h
ttp://www.springframework.org/schema/util"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
      http://www.springframework.org/schema/util
      http://www.springframework.org/schema/util/spring-util-
3.0.xsd">

      <beanclass="com.app.Project"name="proj3">
            <propertyname="projId"value="230"/>
            <propertyname="projMode"value="A3"/>
      </bean>
      <beanclass="com.app.Address"name="addr3">
            <propertyname="addrId"value="190"/>
            <propertyname="loc"value="HYD5"/>
      </bean>
```

```
<util:mapid="sampleMap"map-class="java.util.TreeMap">
	<entry>
		<key>
		<beanclass="com.app.Project"name="proj1">
			<propertyname="projId"value="210"/>
			<propertyname="projMode"value="A1"/>
			</bean>
		</key>
		<beanclass="com.app.Project"name="proj2">
			<propertyname="projId"value="220"/>
			<propertyname="projMode"value="A2"/>
		</bean>
	</entry>
	<entrykey-ref="proj3">
		<refbean="addr3"/>
	</entry>
</util:map>
<beanclass="com.app.Employee"name="empObj">
	<propertyname="empMap">
		<refbean="sampleMap"/>
	</property>
</bean>
</beans>
```

Spring Constructor Dependency Injection:

In Spring Object creation can be done by using default or parameterized constructor.

By default <bean> tag calls default constructor, to specify a parameterized constructor use <constructor-arg> in <bean> tag. One <constructor-arg> indicates one attribute. For example if we want to call 3 parameterized constructor then we have to write <constructor-arg> 3 times. To use feature, Bean class must have parameterized constructors.

Ex:

```
packagecom.app.module;

publicclass Employee {
	privateintempId;
	private String empName;
	privatedoubleempSal;
	public Employee() {
		super();
```

```
	}
	public Employee(intempId, doubleempSal) {
		super();
		this.empId = empId;
		this.empSal = empSal;
	}
	public Employee(intempId, String empName) {
		super();
		this.empId = empId;
		this.empName = empName;
	}
	public Employee(String empName, doubleempSal) {
		super();
		this.empName = empName;
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
				+ ", empSal=" + empSal + "]";
	}

}
```

Spring Configuration File:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:p="http://www.springframework.org/schema/p"xmlns:util="h
ttp://www.springframework.org/schema/util"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
	http://www.springframework.org/schema/util
	http://www.springframework.org/schema/util/spring-util-
3.0.xsd">


	<beanclass="com.app.module.Employee"name="empObj">
	<!-- calls two parameterized constructor -->
		<constructor-arg>
			<value>10</value>
		</constructor-arg>
		<constructor-arg>
			<value>10.32</value>
		</constructor-arg>

	</bean>
```

```
</beans>
```

Test.java:

```
Public class Main{

publicstaticvoid main(String[] args) throws Exception, Exception  {

            ApplicationContext
context=newClassPathXmlApplicationContext("config.xml");
            Employee emp=(Employee)context.getBean("empObj");
            System.out.println(emp);

      }

}
```

Output:

```
Employee [empId=10, empName=null, empSal=10.32]
```

Note: By default <value> tag in constructor tag represents String type. If not found the String type then it choose the nearest one from given constructors in the given order.

For ex:

```
<constructor-arg>
      <value>10</value>
</constructor-arg>
<constructor-arg>
      <value>10.32</value>
</constructor-arg>
```

It will search for String,String . If not then Compare with First constructor of Class, if not matched then second like go on to next constructor.

Note: To specify the particular constructor use the attributes like’

1) type (constructor argument data type)

```
<constructor-argtype="int">
                  <value>10</value>
</constructor-arg>
<constructor-argtype="double">
            <value>10.32</value>
</constructor-arg>
```

2) index (starts from zero)

```
<constructor-argindex="0">
                  <value>10</value>
```

```
   </constructor-arg>
   <constructor-argindex="1">
           <value>10.32</value>
   </constructor-arg>
```

3) name (constructor argument name)

```
<constructor-argname="empId">
           <value>10</value>
</constructor-arg>
<constructor-argname="empSal">
           <value>10.32</value>
</constructor-arg>
```

Note: <constructor-arg> tag also supports writing of code formats which are supported by <property> tag. Like inner bean, ref tag and attribute, Stand-alone collection reference etc..

Ex:

```
<constructor-arg>
                <value>10</value>
          </constructor-arg>
          <constructor-argvalue="ABCD"/>
          <constructor-arg>
                <map>
                      <entrykey="abcd"value="empObj1"/>
                </map>
          </constructor-arg>
          <constructor-arg><!--Innder Bean -->
                <beanclass="com.app.Address">
                      <propertyname="addrId"value="10101"/>
                </bean>
          </constructor-arg>
          <constructor-arg>
                <refbean="addrObj"/>
          </constructor-arg>
          <constructor-arg>
                <props>
                      <propkey="K1">V1</prop>
                      <propkey="K2">V2</prop>
                </props>
          </constructor-arg>
```

While Executing the Constructor, If matching's not found for Data type then it follows 3 Rules

1) **Primitive Type Conversion(Priority 1)**
2) **Auto Boxing And up-casting (Priority 2)**
3) **Var-args (In up casted Order)(Priority 3)**

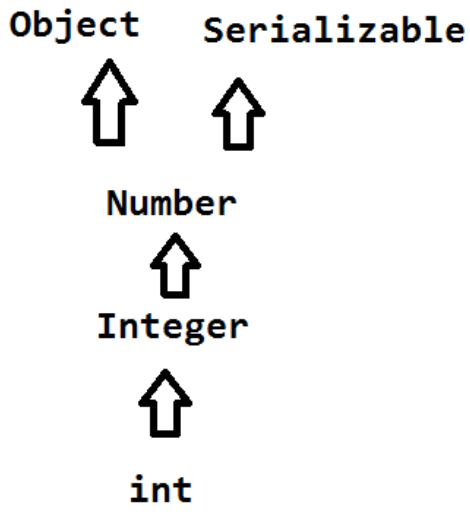
In the Below Example Order is given as:

```
package com.app.module;

import java.io.Serializable;

publicclass Employee {
	public Employee(byte x,byte y) {
		System.out.println("Order-1(never)");
	}
	public Employee(short x,short y) {
		System.out.println("Order-2(never)");
	}
	public Employee(int x,int y) {
		System.out.println("Order1");
	}
	public Employee(long x,long y) {
		System.out.println("Order2");
	}
	public Employee(float x,float y) {
		System.out.println("Order3");
	}
	public Employee(double x,double y) {
		System.out.println("Order4");
	}
	public Employee(Integer x,Integer y) {
		System.out.println("Order5");
	}
	public Employee(Double x,Double y) {
		System.out.println("Order5(never)");
	}
	public Employee(Number x,Number y) {
		System.out.println("Order6");
	}
	public Employee(Serializable x,Serializable y) {
		System.out.println("Order7");
	}
	public Employee(Object x,Object y) {
		System.out.println("Order8");
	}
	public Employee(Integer... integers) {
		System.out.println("Order9");
	}
	public Employee(Number... integers ) {
		System.out.println("Order10");
	}
	public Employee(Serializable... serializables) {
		System.out.println("Order11");
	}
	public Employee(Object... integers) {
		System.out.println("Order12");
```

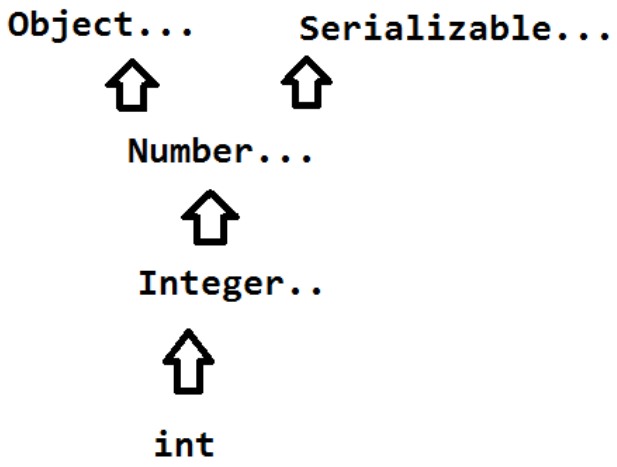
```
	}
	publicstaticvoid main(String[] args) {
		//write the above constructor in any order. They are
called in Specified
		//number formats only.
		Employee emp=new Employee(10,10);
	}
}
```

Process 1:

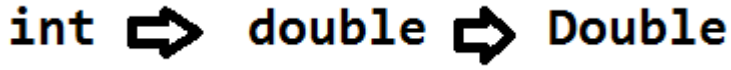
Process 2:



Process 3:



Invalid Process:



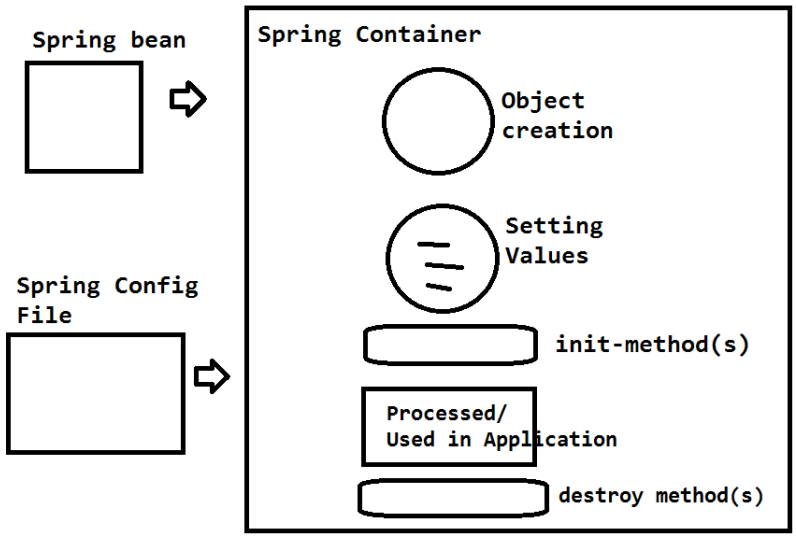
****** Note: at a time type conversion and auto-boxing can never be done.**

Spring Bean Life Cycle Methods:

Spring Bean is having two life cycle methods, which are called automatically by spring container.

1) init-method
2) destroy method

Note: init-method will be executed after creating bean (After CI and SI done). And Destroy method will be called once before the Container shutdown or bean is removed from container.



These are optional methods, an can be configured in 3 ways.

1) Using Interfaces
2) XML Configuration
3) Annotation Based

1) Using Interfaces: In Spring, to configure the init method InitializingBean which is having an abstract method afterPropertiesSet():void and DisposableBean having an abstract method destroy. We need to override these two methods in Bean class.

Ex:

```
packagecom.app.module;

importorg.springframework.beans.factory.DisposableBean;
importorg.springframework.beans.factory.InitializingBean;


publicclass Employee implementsInitializingBean,DisposableBean{
	privateintempId;

	public Employee() {
		super();
		System.out.println("In default constructor");
	}

	publicintgetEmpId() {
		returnempId;
	}

	publicvoidsetEmpId(intempId) {
		this.empId = empId;
		System.out.println("In empId setter method");
	}

	@Override
	public String toString() {
		return"Employee [empId=" + empId + "]";
	}

	@Override
	publicvoidafterPropertiesSet() throws Exception {
		System.out.println("Init -method");

	}
	@Override
	publicvoid destroy() throws Exception {
		System.out.println("Destory Method");
	}
}
```

XML Config Code:-

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:c="http://www.springframework.org/schema/c"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans.xsd">


      <beanclass="com.app.module.Employee"name="empObj">
            <propertyname="empId"value="200"/>
      </bean>
</beans>
```

Test.java

```
publicclassTest{

publicstaticvoid main(String[] args) throws Exception, Exception  {


      AbstractApplicationContextcontext=newClassPathXmlApplicationCo
ntext("config.xml");
            Employee emp=(Employee)context.getBean("empObj");
            System.out.println(emp);
            context.registerShutdownHook();

      }

}
```

2) XML Configuration:- By using <bean> tag specify two attributes, init-method and destroy-method and provide method names for these.

Ex:

```
packagecom.app.module;

publicclass Employee {
      privateintempId;

      public Employee() {
            super();
            System.out.println("In default constructor");
      }

      publicintgetEmpId() {
            returnempId;
      }

      publicvoidsetEmpId(intempId) {
```

```
		this.empId = empId;
		System.out.println("In empId setter method");
	}

	@Override
	public String toString() {
		return"Employee [empId=" + empId + "]";
	}

	publicvoidabcd(){
		System.out.println("In init-method");
	}
	publicvoidmnop(){
		System.out.println("In destroy-method");
	}
}
```

Xml Code:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:c="http://www.springframework.org/schema/c"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd">


	<beanclass="com.app.module.Employee"name="empObj"init-
method="abcd"destroy-method="mnop">
		<propertyname="empId"value="200"/>
	</bean>
</beans>
```

***Test Class Remains Same

3) Annotation Based:- Here two Annotations are used to they are: @PostConstruct (init) and @PreDestroy(destroy)

By Default all Annotations are in spring is disabled mode. We have to enable the annotation before using them, either specific or all annotations.

To Enable these two annotations configure bean class:

org.springframework.context.annotation.CommonAnnotationBeanPostProcessor

Ex:

```
packagecom.app.module;

importjavax.annotation.PostConstruct;
importjavax.annotation.PreDestroy;

publicclass Employee {
	privateintempId;

	public Employee() {
		super();
		System.out.println("In default constructor");
	}

	publicintgetEmpId() {
		returnempId;
	}

	publicvoidsetEmpId(intempId) {
		this.empId = empId;
		System.out.println("In empId setter method");
	}

	@Override
	public String toString() {
		return"Employee [empId=" + empId + "]";
	}

	@PostConstruct
	publicvoidabcd(){
		System.out.println("In init-method");
	}
	@PreDestroy
	publicvoidmnop(){
		System.out.println("In destroy-method");
	}
}
```

XML Code:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:c="http://www.springframework.org/schema/c"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd">
```

```
      <beanclass="org.springframework.context.annotation.CommonAnnot
ationBeanPostProcessor"/>

      <beanclass="com.app.module.Employee" name="empObj">
            <propertyname="empId"value="200"/>
      </bean>
</beans>
```

Note: If we configure 3 ways , then container creates Object and calls 3 init and destroy methods in below order

i) Annotated Methods
ii) Interface Methods
iii) XML Configured Methods

Spring Dependency Check:

All Spring Beans will have dependencies of type Primitive, Collection and References. At the time of creating object injecting values are optional. To make the values injections mandatory enable dependency check. This can be enabled group level (simple, object,all) or individual level (@Required).

This can be written as

1) XML Based (**dependency-check**) (only available in Spring 2.X Versions)
2) Annotation Based **(@Required**)

1) XML Based : here at bean tag level, we have one attribute dependency-check which is having 5 possible values
   a. simple(Primitive type)
   b. objects(Collection and Reference Types)
   c. all
   d. none
   e. default (**Will be decided by default-dependency-check tag**)

If we do not specify any value for the bean it considers as default is the value. But this default value can be one of remaining 4

values and that will be decided by **default-dependency-check attribute from beans tag level.**

Ex:

```
packagecom.app.module;

importjava.util.List;

publicclass Address {

	privateintaddrId;
	private List<String>data;
	publicintgetAddrId() {
		returnaddrId;
	}
	publicvoidsetAddrId(intaddrId) {
		this.addrId = addrId;
	}
	public List<String>getData() {
		returndata;
	}
	publicvoidsetData(List<String> data) {
		this.data = data;
	}
	@Override
	public String toString() {
		return"Address [addrId=" + addrId + ", data=" + data +
"]";
	}

}
```

```
packagecom.app.module;

importjava.util.List;

publicclass Employee {
	privateintempId;
	private String empName;
	private Address addr;
	private List<String>empList;
	publicintgetEmpId() {
		returnempId;
	}
	publicvoidsetEmpId(intempId) {
		this.empId = empId;
	}
```

```
	public String getEmpName() {
		returnempName;
	}
	publicvoidsetEmpName(String empName) {
		this.empName = empName;
	}
	public Address getAddr() {
		returnaddr;
	}
	publicvoidsetAddr(Address addr) {
		this.addr = addr;
	}
	public List<String>getEmpList() {
		returnempList;
	}
	publicvoidsetEmpList(List<String>empList) {
		this.empList = empList;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
+ ", addr="
				+ addr + ", empList=" + empList + "]";
	}

}
```

Configuration file:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:c="http://www.springframework.org/schema/c"xmlns:xsi="ht
tp://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">

	<beanclass="com.app.module.Employee"name="empObj"dependency-
check="simple">

	</bean>
</beans>
```

Main Class:

```
public classMain{

publicstaticvoid main(String[] args) throws Exception, Exception  {
```

```
	ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
		Employee emp=(Employee)context.getBean("empObj");
		System.out.println(emp);

	}

}
```

Output:

```
Exception in thread "main"
org.springframework.beans.factory.UnsatisfiedDependencyException:
Error creating bean with name 'empObj' defined in class path
resource [config.xml]: Unsatisfied dependency expressed through bean
property 'empId': Set this property value or disable dependency
checking for this bean.
```

Note:

a) Here to check all dependencies use all value

```
<beanclass="…"name="…"dependency-check="all">
</bean>
```

b) If we have multiple beans and for all we want to enable by default as simple type then specify the default-dependency-check as simple at bean tag level. Even you can override at specific bean level.

Ex:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:c="ttp://www.springframework.org/schema/c"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd"
	default-dependency-check="simple"
	>

<beanclass="com.app.module.Employee"name="empObj">
	<!-- here it takes default dependency check value simple as
default value -->

	</bean>
<beanclass="com.app.module.Address"name="addrObj"dependency-
check="default">
	<!-- it is going to check default value. here  default is
simple-->
```

```
</bean>
<beanclass="com.app.module.Project"name="projObj"dependency-
check="simple">
<!-- it overrides the default value to simple only for this Project
Bean -->
      </bean>
      </beans>
```

---

2) Using @Required Annotation:

By default all annotations are in disabled mode in spring, to activate this in spring configure the class RequiredAnnotationBeanPostProcessor. Or use Context schema, **<context:annotation-config/>**

@Required annotation is applicable at setter method level. It is used to specify the particular fields from the all dependencies. For example like 1 Primitive, 1 Reference from 10 Dependencies.

Ex:

```
packagecom.app.module;

importjava.util.List;

importorg.springframework.beans.factory.annotation.Required;

publicclass Employee {
      privateintempId;
      private String empName;
      private Address addr;
      private List<String>empList;
      publicintgetEmpId() {
            returnempId;
      }
      @Required
      publicvoidsetEmpId(intempId) {
            this.empId = empId;
      }
      public String getEmpName() {
            returnempName;
      }
      publicvoidsetEmpName(String empName) {
            this.empName = empName;
      }
      public Address getAddr() {
            returnaddr;
      }
      publicvoidsetAddr(Address addr) {
```

```
		this.addr = addr;
	}
	public List<String>getEmpList() {
		returnempList;
	}
	publicvoidsetEmpList(List<String>empList) {
		this.empList = empList;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
+ ", addr="
				+ addr + ", empList=" + empList + "]";
	}

}
```

Format 1)

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:c="http://www.springframework.org/schema/c"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd">

	<beanclass="org.springframework.beans.factory.annotation.Requi
redAnnotationBeanPostProcessor">

	</bean>
	<beanclass="com.app.module.Employee"name="empObj">


	</bean>

</beans>
```

Format 2)

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/context
```

```
      http://www.springframework.org/schema/context/spring-
context.xsd">

      <context:annotation-config/>
      <!-- It activates all spring annotations -->
      <beanclass="com.app.module.Employee"name="empObj">
      </bean>

</beans>
```

Test Class

```
public class Main{

publicstaticvoid main(String[] args) throws Exception, Exception  {


      ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
            Employee emp=(Employee)context.getBean("empObj");
            System.out.println(emp);

      }

}
```

Output:

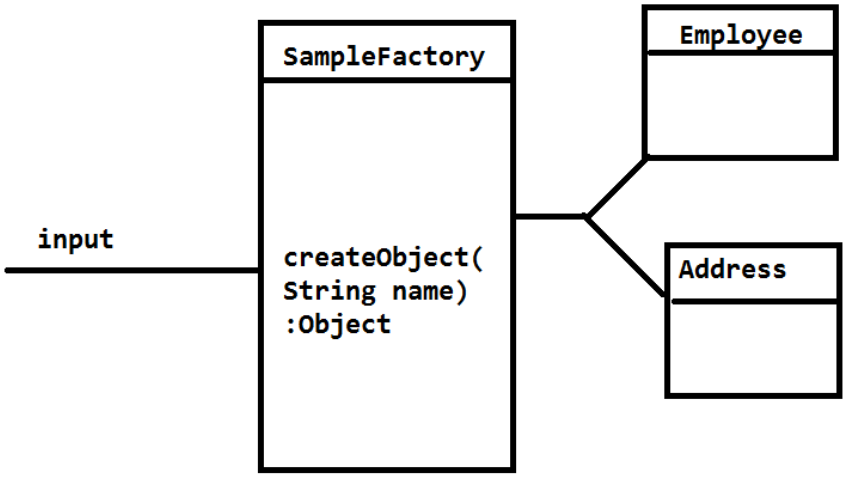
```
Exception in thread "main"
org.springframework.beans.factory.BeanCreationException: Error
creating bean with name 'empObj' defined in class path resource
[config.xml]: Initialization of bean failed; nested exception is
org.springframework.beans.factory.BeanInitializationException:
Property 'empId' is required for bean 'empObj'
```


Spring Instance Factory and Static Factory Design pattern:-

Spring supports in-built factory design pattern in 2 ways instance factory and static factory.

Here factory method returns object type. It will take dynamic input based on that, it will create the object. In this case we need to configure only Factory class in spring configuration file.

Ex:



Code:

```
packagecom.app.module;

publicclass Employee {
	privateintempId;
	private String empName;
	public Employee() {
		super();
	}
	public Employee(intempId, String empName) {
		super();
		this.empId = empId;
		this.empName = empName;
	}
	publicintgetEmpId() {
		returnempId;
	}
	publicvoidsetEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoidsetEmpName(String empName) {
		this.empName = empName;
```

```
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
+ "]";
	}


}
```

```
packagecom.app.module;

importjava.util.List;

publicclass Address {

	privateintaddrId;
	private String loc;
	public Address() {
		super();
	}
	public Address(intaddrId, String loc) {
		super();
		this.addrId = addrId;
		this.loc = loc;
	}
	publicintgetAddrId() {
		returnaddrId;
	}
	publicvoidsetAddrId(intaddrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		returnloc;
	}
	publicvoidsetLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return"Address [addrId=" + addrId + ", loc=" + loc + "]";
	}

}
```

```
packagecom.app.module;

publicclassSimpleFactory {

	publicstatic Object createObject(String name){
		if("emp".equalsIgnoreCase(name)){
```

```
                returnnew Employee(10,"ABCD");
          }elseif("addr".equalsIgnoreCase(name)){
                returnnew Address(200, "HYD");
          }
          returnnull;
     }
}
```

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
     xmlns:context="http://www.springframework.org/schema/context"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/bean
s
     http://www.springframework.org/schema/beans/spring-beans.xsd">
     <!--
     Java code :
     Object sf=SampleFactory.createObject("emp");
      -->
     <beanclass="com.app.module.SimpleFactory"name="sf"factory-
method="createObject">
          <constructor-arg>
               <value>addr</value>
          </constructor-arg>
     </bean>

</beans>
```

Test.java

```
publicstaticvoid main(String[] args) throws Exception, Exception  {


     ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
          Object ob=context.getBean("sf");
          if(obinstanceof Employee){
               Employee emp=(Employee)ob;
               System.out.println(emp);
          }elseif(obinstanceof Address){
               Address add=(Address)ob;
               System.out.println(add);
          }
          System.out.println("done");
     }
```

Output:

```
Address [addrId=200, loc=HYD]
Done
```

For Instance factory changed cod is:

Factory class:

```
packagecom.app.module;

publicclassSimpleFactory {

	public Object createObject(String name){
		if("emp".equalsIgnoreCase(name)){
			returnnew Employee(10,"ABCD");
		}elseif("addr".equalsIgnoreCase(name)){
			returnnew Address(200, "HYD");
		}
		returnnull;
	}
}
```

XML Code:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd">
	<!--
	Java code :
	SimpleFactorysf=new SimpleFactory();
	 -->
	<beanclass="com.app.module.SimpleFactory" name="sf1">

	</bean>
	<!-- Object sf=sf1.createObject("emp"); -->
	<beanname="sf"factory-bean="sf1"factory-method="createObject">
		<constructor-arg>
			<value>emp</value>
		</constructor-arg>
	</bean>
</beans>
```

Output:

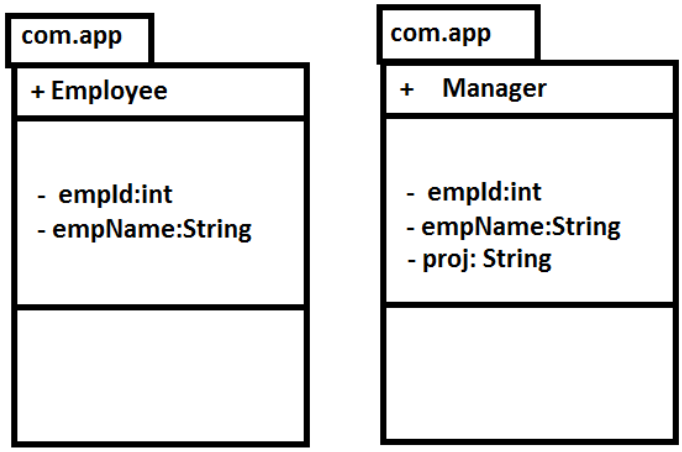
```
Employee [empId=10, empName=ABCD]
done
```

Spring Bean Inheritance and Abstraction:

In **Spring inheritance** indicates reusability of Dependency code written in Spring configuration file (XML), but not the meaning of Java Style Inheritance concept.

Abstraction indicates stopping of creating a bean (object) in Spring container, normally every bean can be abstract even though it's class is not abstract. But basically this is used to specify the common properties as abstract bean which contains only name and abstraction (true) with some commons Dependency Injections.

Ex: for Inheritance : to specify the reading of parent bean data ,use **parent** attribute at bean tag level.



java Code:

```
package com.app;

public class Employee{

	private int empId;
	private String empName;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
```

```
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" +
empName + "]";
	}


}
=======

package com.app;

public class Manager {

	private int empId;
	private String empName;
	private String proj;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public String getProj() {
		return proj;
	}
	public void setProj(String proj) {
		this.proj = proj;
	}
	@Override
	public String toString() {
		return "Manager [empId=" + empId + ", empName=" + empName
+ ", proj="
				+ proj + "]";
	}

}


=======

Config.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/bean
s

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

      <bean class="com.app.Employee" name="empObj">
            <property name="empId" value="100"/>
            <property name="empName" value="ABCD"/>
      </bean>

      <bean class="com.app.Manager" name="mObj" parent="empObj">
            <property name="proj" value="XYZ"/>
      </bean>

</beans>
```

=====

Test.java

```
package com.app;

import
org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

      public static void main(String[] args) {
            AbstractApplicationContext context = new
ClassPathXmlApplicationContext("config.xml");
            Object obj = context.getBean("mObj");
            System.out.println(obj);

      }
}
```

======

output:

```
Manager [empId=100, empName=ABCD, proj=XYZ]
```

Note 1): Even we can use a common bean as template for both the classes common data representation.

Format 1)

```
<bean name="commonData" abstract="true">
            <property name="empId" value="100"/>
            <property name="empName" value="ABCD"/>
      </bean>
            <bean class="com.app.Employee" name="empObj"
      parent="commonData">
      </bean>

      <bean class="com.app.Manager" name="mObj" parent="empObj">
            <property name="proj" value="XYZ"/>
      </bean>
```

format 2)

```
<bean name="commonData" abstract="true">
            <property name="empId" value="100"/>
            <property name="empName" value="ABCD"/>
      </bean>
      <bean class="com.app.Employee" name="empObj"
parent="commonData">
      </bean>

      <bean class="com.app.Manager" name="mObj" parent="commonData">
            <property name="proj" value="XYZ"/>
      </bean>
```

Note 2): If value given by parent is not satisfied by child bean, then child bean can override the value by performing dependency injection again.

```
<bean name="commonData" abstract="true">
            <property name="empId" value="100"/>
            <property name="empName" value="ABCD"/>
      </bean>
      <bean class="com.app.Employee" name="empObj"
parent="commonData">
      </bean>

      <bean class="com.app.Manager" name="mObj" parent="commonData">
            <property name="empId" value="250"/>
            <property name="proj" value="XYZ"/>
      </bean>
```

Spring Auto wiring:-

Autowiring is a concept of injecting a spring bean automatically without writing <ref/> tag by programmer. Programmer does not required to write explicitly Dependency Injection.

Autowiring can be configured in 2 ways :

1) XML based

2) Annotation Based

1) XML Based: XML based autowiring can be classified as

a) byName : compares the spring bean (java code) filed name (variable name) and configuration code (XML file) bean name (bean tag name) , if they are matched then automatically those objects will b bonded with each other using setter injection.

b) byType: It compares bean class variable Data type and spring bean class type. If both are matched then it will do setter injection.

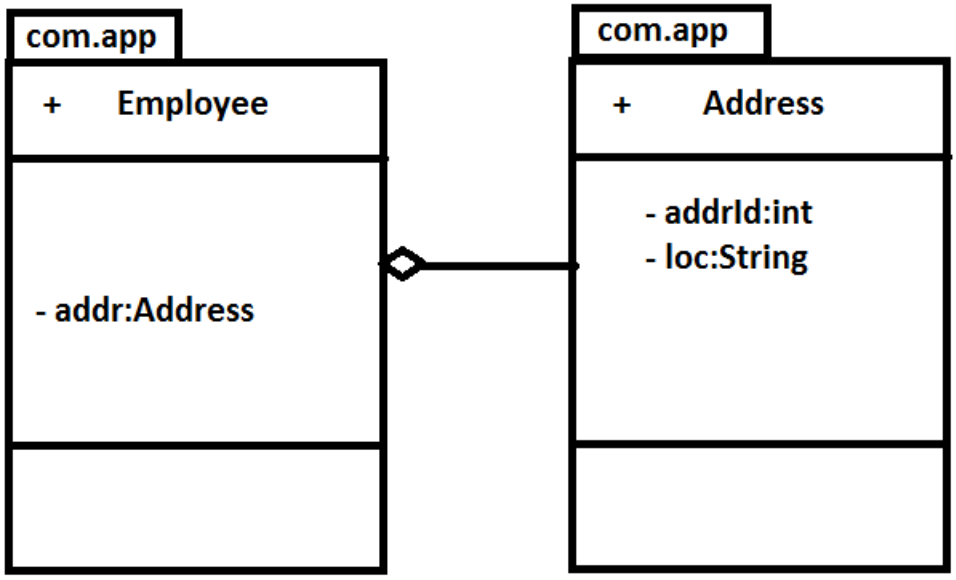
c) constructor: It check for parameterized constructor for creating object with reference type as parameter. If not found then uses default constructor at last. Always checks More number of parameters first, if not matched then next level less no of parameters constructor will be compared, and then goes on up to default constructor.

d) no : it will not do any auto wiring. It is disabled by default.

e) default (default value is no, even we can change this)

f) autodetect (works only in older versions like 2.X) : It works like byType if default constructor is available in spring bean, if not works like constructor if parameterized constructor is available.

ex: consider the below example for above concept. Employee class has a Address class Dependency.



Java code:

```
package com.app;

public class Address {

	private int addrId;
	private String loc;

	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}


}
=====
```

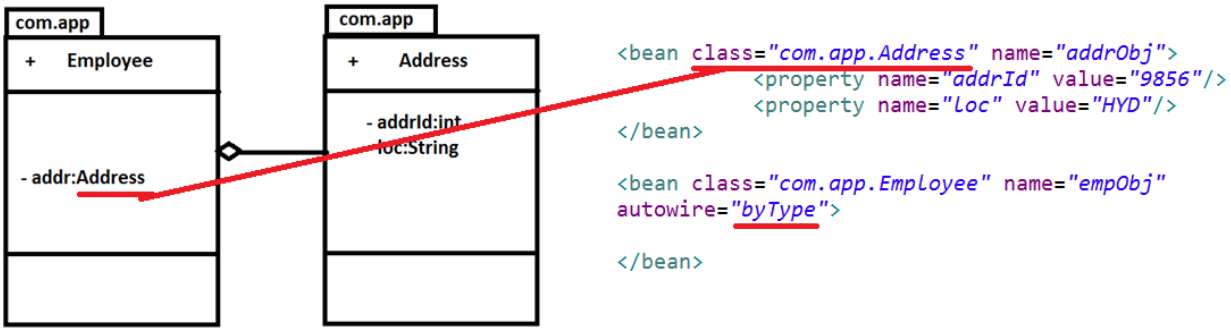
```
package com.app;

public class Employee{

	private Address addr;

	public Address getAddr() {
		return addr;
	}

	public void setAddr(Address addr) {
		this.addr = addr;
	}

	@Override
	public String toString() {
		return "Employee [addr=" + addr + "]";
	}


}
```

======

code byType:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:util="http://www.springframework.org/schema/util"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

	http://www.springframework.org/schema/util

	http://www.springframework.org/schema/util/spring-util.xsd

	http://www.springframework.org/schema/context

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

	<bean class="com.app.Address" name="addrObj">
		<property name="addrId" value="9856"/>
		<property name="loc" value="HYD"/>
```

```
	</bean>

	<bean class="com.app.Employee" name="empObj"
autowire="byType">

	</bean>
</beans>
```

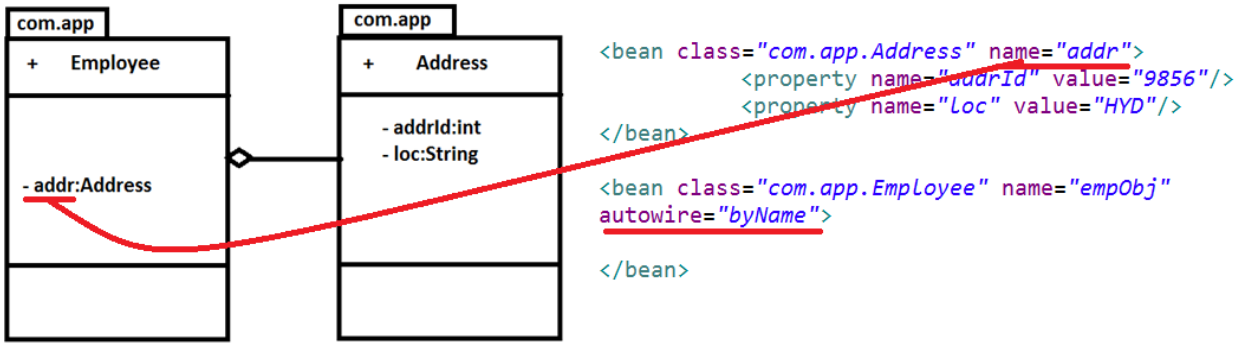


==========

code byName:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:util="http://www.springframework.org/schema/util"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

	http://www.springframework.org/schema/util

	http://www.springframework.org/schema/util/spring-util.xsd

	http://www.springframework.org/schema/context

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

	<bean class="com.app.Address" name="addr">
		<property name="addrId" value="9856"/>
		<property name="loc" value="HYD"/>
	</bean>

	<bean class="com.app.Employee" name="empObj"
autowire="byName">
```

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



========

constructor:

to use this option class must have at least one parameterized constructor.

Always constructor with more parameters will be compared first , if not matched then it goes to next level more parameters constructor (like 4 parameters, 3 parameters ..) until zero/default constructor. Even If default constructor not found then spring container throws an exception saying that no constructor found.

Java Code:

```
package com.app;

public class Employee{

	private Address addr;


	public Employee() {
		super();
		System.out.println("In default");
	}

	public Employee(Address addr) {
		super();
		this.addr = addr;
		System.out.println("in Param");
	}

	public Address getAddr() {
		return addr;
	}

	public void setAddr(Address addr) {
```

```
			this.addr = addr;
		}

		@Override
		public String toString() {
			return "Employee [addr=" + addr + "]";
		}


}
====

package com.app;

public class Address {

		private int addrId;
		private String loc;

		public int getAddrId() {
			return addrId;
		}
		public void setAddrId(int addrId) {
			this.addrId = addrId;
		}
		public String getLoc() {
			return loc;
		}
		public void setLoc(String loc) {
			this.loc = loc;
		}
		@Override
		public String toString() {
			return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
		}


}

package com.app;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
public static void main(String[] args) throws Exception, Exception
{
```

```
            ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
            Address address=context.getBean("addrObj",Address.class);
            System.out.println(address);
            Employee
employee=context.getBean("empObj",Employee.class);
            System.out.println(employee);

            System.out.println("done");
      }


}
  ====
```

XML Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s

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

      http://www.springframework.org/schema/util

      http://www.springframework.org/schema/util/spring-util.xsd

      http://www.springframework.org/schema/context

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

      <bean class="com.app.Address" name="addr">
            <property name="addrId" value="9856"/>
            <property name="loc" value="HYD"/>
      </bean>

      <bean class="com.app.Employee" name="empObj"
autowire="constructor">

      </bean>
</beans>

===========
```

```
package com.app;

import
org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

	public static void main(String[] args) {
		AbstractApplicationContext context = new
ClassPathXmlApplicationContext("config.xml");
		Employee obj = (Employee)context.getBean("empObj");
		System.out.println(obj);


	}
}
======
```

Note: 1) If more than one matchi9ng found while doing constructor , then it compares names to inject the bean: for ex:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:util="http://www.springframework.org/schema/util"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/util
	http://www.springframework.org/schema/util/spring-util.xsd
	http://www.springframework.org/schema/context
	http://www.springframework.org/schema/context/spring-
context.xsd
                                    ">

	<bean class="com.app.Address" name="addr">
		<property name="addrId" value="956"/>
		<property name="loc" value="HYD"/>
	</bean>
	<bean class="com.app.Address" name="addr2">
		<property name="addrId" value="985"/>
		<property name="loc" value="HYD1"/>
	</bean>
	<bean class="com.app.Employee" name="empObj"
autowire="constructor">

	</bean>
```

```
</beans>
```

in the above code addr,addr2 are two objects then it compares the names to inject. So **addr** will be select to inject. If no name matched then no object will be injected.

=========

autodetect:

Java code:

```
package com.app;

public class Address {

	private int addrId;
	private String loc;

	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}


}
```

============

```
package com.app;

public class Employee{

	private Address addr;


	public Employee() {
		super();
		System.out.println("In default");
```

```
	}

	public Employee(Address addr) {
		super();
		this.addr = addr;
		System.out.println("in Param");
	}

	public Address getAddr() {
		return addr;
	}

	public void setAddr(Address addr) {
		this.addr = addr;
		System.out.println("in setter");
	}

	@Override
	public String toString() {
		return "Employee [addr=" + addr + "]";
	}


}
===

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:util="http://www.springframework.org/schema/util"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s

	http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd

	http://www.springframework.org/schema/util

	http://www.springframework.org/schema/util/spring-util.xsd

	http://www.springframework.org/schema/context

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

	<bean class="com.app.Address" name="addr">
		<property name="addrId" value="9"/>
```

```
            <property name="loc" value="HYD9"/>
      </bean>
      <bean class="com.app.Employee" name="empObj"
autowire="autodetect">

      </bean>
</beans>
```

```
package com.app;

import
org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

      public static void main(String[] args) {
            AbstractApplicationContext context = new
ClassPathXmlApplicationContext("config.xml");
            Employee obj = (Employee)context.getBean("empObj");
            System.out.println(obj);

      }
}
```

Note: if you execute the above code it will check for default constructor and if found it performs byType Autowiring. If you comment default constructor then it uses parameterized constructor.

output:

```
In default
in setter
Employee [addr=Address [addrId=9, loc=HYD9]]
```

on comment of default constructor:

```
/*public Employee() {
            super();
            System.out.println("In default");
}*/
```

output:

```
in Param
Employee [addr=Address [addrId=9, loc=HYD9]]
```

Spring Bean Scope:

1) singleton : This scope specifies , for every spring bean and for one configuration spring container creates one object. This is the default scope of all spring beans.

2) prototype: This scope indicates, creating a spring bean object on every read in spring container.

3) request: It is applicable to Web applications only. On every request sent by client it creates object.

4) session: it is also related to web applications. On every session, creates new object in container.

5)global session:- it is related to portlets concept. It creates a spring bean for the global session of portlet.

To specify the spring bean scope use bean tag scope attribute.

ex:

```
package com.app;

public class Employee{

	private int empId;
	private String empName;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" +
empName + "]";
	}

}
```

```
<bean class="com.app.Employee" name="empObj" scope="singleton">
            <property name="empId" value="885"/>
            <property name="empName" value="ABCD"/>
</bean>
```

```
package com.app;

import
org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

      public static void main(String[] args) {
            AbstractApplicationContext context = new
ClassPathXmlApplicationContext("config.xml");
            Employee obj = (Employee)context.getBean("empObj");
            System.out.println(obj.hashCode());
            Employee obj2 = (Employee)context.getBean("empObj");
            System.out.println(obj2.hashCode());
            Employee obj3 = (Employee)context.getBean("empObj");
            System.out.println(obj3.hashCode());

      }
}
 retusn output of same hash code for all above objects.
```

========

to specify other scopes:

```
<bean class="..." name="..." scope="prototype">... </bean>

<bean class="..." name="..." scope="request">... </bean>

<bean class="..." name="..." scope="session">... </bean>
```

============

Lookup method Injection:-

If a independent class is prototype scope and dependent class is single tone scope then always spring container returns first time banded object with sinleton class , even though it creates new object for prototype.

To change this functionality use, look up method injection, for this we need to add CGLIB Jar.

download link : (control button +mouse click to download)

http://central.maven.org/maven2/cglib/cglib/3.1/cglib-3.1.jar

Specify the abstract method that should return the Prototype class and call that method somewhere in the java code. make class as even abstract, that class will be implemented as proxy by CGLIB jar with same name.

ex:

java code:

```
package com.app;

public class Address {

	private int addrId;
	private String loc;

	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc +
"]";
	}


}
```

=====

```
package com.app;

public abstract class Employee{

	private Address addr;
```

```
	public Address getAddr() {
		addr=createObject();
		return addr;
	}

	@Override
	public String toString() {
		return "Employee [addr=" + addr + "]";
	}

	protected abstract Address createObject();

}
```

====

config.xml

```
                              ">
	<bean class="com.app.Address" name="addrObj"
scope="prototype">
		<property name="addrId" value="152"/>
		<property name="loc" value="HYD"/>
	</bean>
	<bean class="com.app.Employee" name="empObj"
scope="singleton">
		<lookup-method bean="addrObj" name="createObject"/>
	</bean>
```

=======

Test.java

```
package com.app;

import
org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

	public static void main(String[] args) {
		AbstractApplicationContext context = new
ClassPathXmlApplicationContext("config.xml");
Employee obj = (Employee)context.getBean("empObj");
System.out.println("emp:"+obj.hashCode()+",addr:"+obj.getAddr().hash
Code());
Employee obj2 = (Employee)context.getBean("empObj");
System.out.println("emp:"+obj2.hashCode()+",addr:"+obj2.getAddr().ha
shCode());
Employee obj3 = (Employee)context.getBean("empObj");
```

```
System.out.println("emp:"+obj3.hashCode()+",addr:"+obj3.getAddr().ha
shCode());
Employee obj4 = (Employee)context.getBean("empObj");
System.out.println("emp:"+obj4.hashCode()+",addr:"+obj4.getAddr().ha
shCode());

      }
}
output:

emp:5122060,addr:13177628
emp:5122060,addr:17152415
emp:5122060,addr:13508999
emp:5122060,addr:16471729
```

Stereo type Annotations in spring:-

These are the annotations used to create a bean (object) in spring container, without any XML configuration. @Component is the Stereo type Annotation. And even @Controller, @Repository, @Service are also layer level Stereo type annotations.

To activate this annotation we need to specify the annotation activation and base-package for component scan in Spring Configuration file.

Ex:

```
packagecom.app.module;

importorg.springframework.stereotype.Component;

@Component("addrObj")
publicclass Address {


}
```

Equal XML Code is <bean class=”com.app.module.Address” name=”addrObj”/>

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-
context.xsd">

      <context:annotation-config/>
      <context:component-scanbase-package="com.app"/>

</beans>
```

```
packagecom.app;

importorg.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

importcom.app.module.Address;

publicclass Test {

publicstaticvoid main(String[] args) throws Exception, Exception  {


      ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
            Address cons =context.getBean("addrObj", Address.class);
            System.out.println(cons);
      }
}
```

Output:

```
com.app.module.Address@4c4975
```

@Component if not having any parameter value then it takes class name as bean name by converting first letter of class name as lower case.

Ex:

```
@Component
publicclass Address {

}
```

For this equal XML Code is

```
<bean class=”Address” name=”address”/>
```

@Value:-This is used to inject a value using Annotation to a primitive dependency.

@Value can be specified as direct value, Expression value and dynamic value.

Ex:

```
packagecom.app.module;

importorg.springframework.beans.factory.annotation.Value;
importorg.springframework.stereotype.Component;

@Component("addrObj")
publicclass Address {

	@Value("350")
	privateintempId;
	@Value("#{(new java.util.Random()).nextDouble()}")
	privatedoubleempSal;
	@Value("#{32>99?true:false}")
	privatebooleanstatus;
	@Override
	public String toString() {
		return"Address [empId=" + empId + ", empSal=" + empSal +
", status="
				+ status + "]";
	}

}
```

XML Code:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/context
	http://www.springframework.org/schema/context/spring-
context.xsd">

	<context:annotation-config/>
	<context:component-scanbase-package="com.app"/>

</beans>
```

Test.java

```
packagecom.app;
```

```
importorg.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

importcom.app.module.Address;

publicclass Test {

publicstaticvoid main(String[] args) throws Exception, Exception  {


      ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
            Address cons =context.getBean("addrObj", Address.class);
            System.out.println(cons);
      }
}
```

Bean Externalization : Loading value from .properties file and assigning values to a particular bean, in known as bean externalization.

To Create a properties file: right click on src-> new -> other -> file-> next-> provider any name (like jdbc.properties)

It maintains data in the form of key value pair. Both key and value should be in string format. Key is case sensitive (num ,Num are different). If a key is used to more than one time in properties file then last value will be considered to the key. To read a key value use below syntax :
${key-name}.

Ex:

```
packagecom.app.module;

publicclass Employee {
      privateintempId;
      privatedoubleempSal;
      privatebooleanstatus;
      publicintgetEmpId() {
            returnempId;
      }
      publicvoidsetEmpId(intempId) {
            this.empId = empId;
      }
      publicdoublegetEmpSal() {
            returnempSal;
```

```
	}
	publicvoidsetEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	publicbooleanisStatus() {
		returnstatus;
	}
	publicvoidsetStatus(boolean status) {
		this.status = status;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empSal=" + empSal +
", status="
				+ status + "]";
	}

}
```

abcd.properties:

```
#This is a comment line in  Properties file
idValue=250
sal=96369.36
stat=true
```

XML File:

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/context
	http://www.springframework.org/schema/context/spring-
context.xsd">

	<context:property-placeholder location="abcd.properties"/>

	<beanclass="com.app.module.Employee" name="empObj">
		<propertyname="empId">
			<value>${idValue}</value>
		</property>
		<propertyname="empSal">
			<value>${sal}</value>
		</property>
		<propertyname="status">
			<value>${stat}</value>
		</property>
```

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

Test.java

```
packagecom.app;

importorg.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

importcom.app.module.Employee;

publicclass Test {

publicstaticvoid main(String[] args) throws Exception, Exception  {


	ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
		Employee cons =context.getBean("empObj", Employee.class);
		System.out.println(cons);
	}
}
```

Output:

```
Employee [empId=250, empSal=96369.36, status=true]
```

Note: if Key name does not exist then spring container throws Exceptionorg.springframework.beans.factory.BeanDefinitionStoreException: Invalid bean definition: Could not resolve placeholder 'idValue1' in string value "${idValue1}"

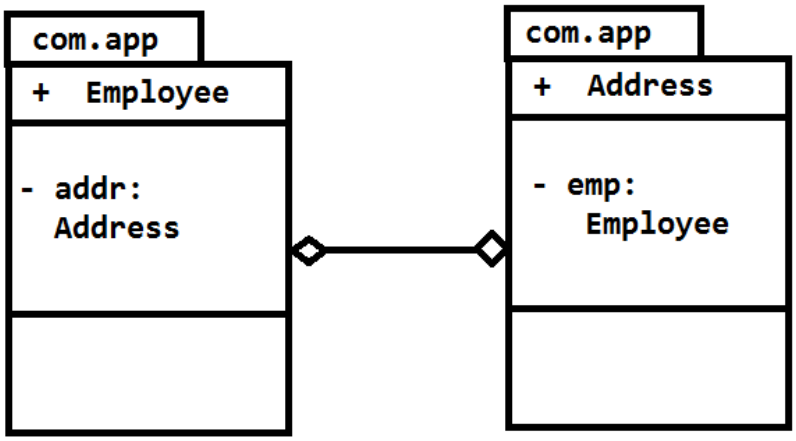
Spring Circular Dependency:

If two classes are dependent on each other, then creating object will be very difficult. In this case spring provides best solution.

First it creates objects using default constructor then it performs injections.

Example:

Consider Employee and Address are the two classes which are having dependency on each other.



Java code is:

```
packagecom.app;
publicclass Employee {
	private Address addr;

	public Employee() {
		super();
		System.out.println("In Employee Default Constructor");
	}

	public Address getAddr() {
		returnaddr;
	}

	publicvoidsetAddr(Address addr) {
		this.addr = addr;
		System.out.println("In Employee class , Address setter");
	}


}
```

```
packagecom.app;

publicclass Address {
	private Employee emp;

	public Address() {
		super();
		System.out.println("In Address Default Constructor");
	}

	public Employee getEmp() {
		returnemp;
	}

	publicvoidsetEmp(Employee emp) {
		this.emp = emp;
		System.out.println("In Address class , Employee setter");
	}



}
```

Config.xml

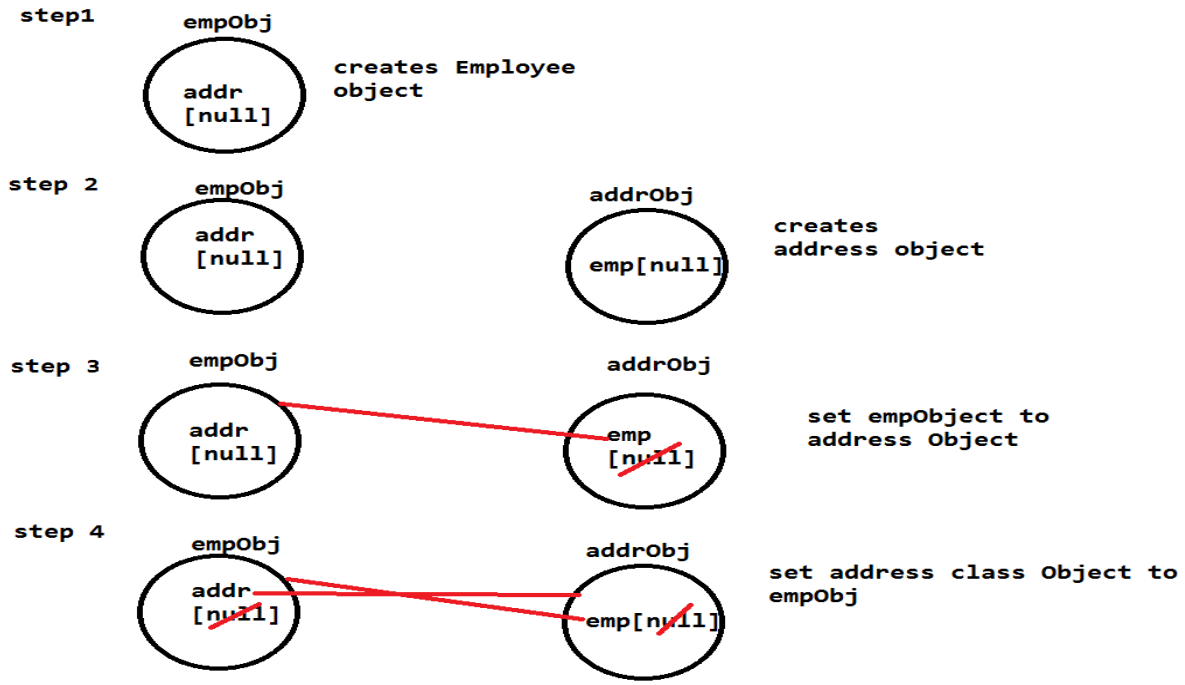
```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s
	http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/context
	http://www.springframework.org/schema/context/spring-
context.xsd">

	<beanclass="com.app.Employee"name="empObj">
		<propertyname="addr">
			<refbean="addrObj"/>
		</property>
	</bean>
	<beanclass="com.app.Address"name="addrObj">
		<propertyname="emp">
			<refbean="empObj"/>
		</property>
	</bean>
</beans>
```

```
packagecom.app;

importorg.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

importcom.app.Employee;

publicclass Test {

publicstaticvoid main(String[] args) throws Exception, Exception  {


      ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");
            Employee cons =context.getBean("empObj", Employee.class);
            System.out.println(cons);
      }
}
```

Output: In Employee Default Constructor

```
In Address Default Constructor
In Address class , Employee setter
In Employee class , Address setter
com.app.Employee@a761fe
```

Step by Step Process



Spring -JDBC Template Sample Code:

Spring-JDBC:

- Automatic connection handling.
- Template Based programming
- Multiple Data source availability
- Built-in Methods for SQL operations.
- RowMapper for Multi object Operations.

Config.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

http://www.springframework.org/schema/util

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

      <bean
class="org.springframework.jdbc.datasource.DriverManagerDataSource"
name="dataSource">
            <property name="driverClassName" value=""/>
            <property name="url" value=""/>
            <property name="username" value=""/>

      </bean>
      <bean class="org.springframework.jdbc.core.JdbcTemplate"
name="jdbcTemplateObj">
            <property name="dataSource">
                  <ref bean="dataSource"/>
            </property>
      </bean>
      <bean class="com.app.EmployeeDaoImpl" name="empDao">
            <property name="template">
                  <ref bean="jdbcTemplateObj"/>
            </property>
      </bean>
```

```
</beans>
```

Employee.java

```
package com.app;


public class Employee {

	private Integer empId;
	private String empName;
	private Double empSal;

	public Employee() {
		super();
	}


	public Employee(Integer empId, String empName, Double empSal)
{
		super();
		this.empId = empId;
		this.empName = empName;
		this.empSal = empSal;
	}


	public Integer getEmpId() {
		return empId;
	}
	public void setEmpId(Integer empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public Double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(Double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" +
empName
```

```
                        + ", empSal=" + empSal + "]";
	}



}
```

```
package com.app;


public interface IEmployeeDao {
	public void createEmployee(Employee emp);


}
```

```
package com.app;


import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDaoImpl implements IEmployeeDao {

	private JdbcTemplate template;

	public void setTemplate(JdbcTemplate template) {
		this.template = template;
	}

	@Override
	public void createEmployee(Employee emp) {
		String sql="insert into employee values(?,?,?)";

	template.update(sql,emp.getEmpId(),emp.getEmpName(),emp.getEmp
Sal());

	}


}
```

Main.java

```
package com.app;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;



public class Main {
```

```
	public static void main(String[] args) {

		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		IEmployeeDao dao=(IEmployeeDao)context.getBean("empDao");
		Employee emp=new Employee();

		emp.setEmpId(101);
		emp.setEmpName("abcd");
		emp.setEmpSal(200.36);
		dao.createEmployee(emp);

	}

}
```

Sample Program JDBC
Spring-JDBC-Oracle-Multi-table –operations:
Jars: database Jar(ojdbc14.jar)
Spring JARs + Commons Logging Jar.
Oracle Tables: (create these tables first in oracle)

```
CREATE table "EMP" (
    "EID"        NUMBER,
    "ENAME"      VARCHAR2(50),
    "ESAL"       NUMBER(22,5),
    constraint  "EMP_PK" primary key ("EID")
)
/
CREATE table "ADDR" (
    "AID"        NUMBER,
    "LOC"        VARCHAR2(50),
    "EID"        NUMBER,
    constraint  "ADDR_PK" primary key ("AID")
)
/

ALTER TABLE "ADDR" ADD CONSTRAINT "ADDR_FK"
FOREIGN KEY ("EID")
REFERENCES "EMP" ("EID")

/
```

Java Code:

```
package com.app;

public class Address {
	private int addrId;
	private String loc;
	publicint get AddrId() {
		return addrId;
	}
	Public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		returnloc;
	}
	Public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return"Address [addrId=" + addrId + ", loc=" + loc + "]";
	}
	@Override
	Public int hashCode() {
		Final int prime = 31;
		int result = 1;
		result = prime * result + addrId;
		result = prime * result + ((loc == null) ? 0 :
loc.hashCode());
		return result;
	}
	@Override
	publicboolean equals(Object obj) {
		if (this == obj)
			returntrue;
		if (obj == null)
			returnfalse;
		if (getClass() != obj.getClass())
			returnfalse;
		Address other = (Address) obj;
		if (addrId != other.addrId)
			returnfalse;
		if (loc == null) {
			if (other.loc != null)
				returnfalse;
		} elseif (!loc.equals(other.loc))
			returnfalse;
		returntrue;
	}}
```

```
package com.app;

publicclass Employee {
	privateintempId;
	private String empName;
	privatedoubleempSal;
	private Address addr;
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	public Address getAddr() {
		returnaddr;
	}
	publicvoid setAddr(Address addr) {
		this.addr = addr;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
				+ ", empSal=" + empSal + ", addr=" + addr +
"]";
	}
	@Override
	publicint hashCode() {
		finalint prime = 31;
		int result = 1;
		result = prime * result + ((addr == null) ? 0 :
addr.hashCode());
		result = prime * result + empId;
		result = prime * result + ((empName == null) ? 0 :
empName.hashCode());
		long temp;
		temp = Double.doubleToLongBits(empSal);
		result = prime * result + (int) (temp ^ (temp >>> 32));
```

```
            return result;
      }
      @Override
      Public boolean equals(Object obj) {
            if (this == obj)
                  return true;
            if (obj == null)
                  return false;
            if (getClass() != obj.getClass())
                  return false;
            Employee other = (Employee) obj;
            if (addr == null) {
                  if (other.addr != null)
                        return false;
            } elseif (!addr.equals(other.addr))
                  returnfalse;
            if (empId != other.empId)
                  returnfalse;
            if (empName == null) {
                  if (other.empName != null)
                        returnfalse;
            } elseif (!empName.equals(other.empName))
                  Return false;
            if (Double.doubleToLongBits(empSal) != Double
                        .doubleToLongBits(other.empSal))
                  returnfalse;
            returntrue;
      }



}
```

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-
context.xsd">
      <context:annotation-config/>
      <context:component-scanbase-package="com.app"/>
      <beanclass="org.springframework.jdbc.datasource.DriverManagerD
ataSource"
            name="dsObj">
```

```
	<propertyname="driverClassName"value="oracle.jdbc.driver.Oracl
eDriver"/>

	<propertyname="url"value="jdbc:oracle:thin:@localhost:1521:xe"
/>
		<propertyname="username"value="system"/>
		<propertyname="password"value="system"/>
	</bean>
	<beanclass="org.springframework.jdbc.core.JdbcTemplate"name="t
empObj">
		<propertyname="dataSource">
			<refbean="dsObj"/>
		</property>
	</bean>


</beans>
```

```
package com.app;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Component;

@Component("empDao")
publicclass EmployeeDaoImpl {

	@Autowired
	private JdbcTemplate template;

	publicvoid insertDataToDB(Employee employee){
		String sql="insert into emp values
(?,?,?)";//eid(PK),ename,esal
		String sql2="insert into addr values
(?,?,?)";//aid(PK),loc,eid(FK)
		int
i=template.update(sql,employee.getEmpId(),employee.getEmpName(),empl
oyee.getEmpSal());
		int i1=-1;
		if(i==1){
			Address addr=employee.getAddr();
```

```
		i1=template.update(sql2,addr.getAddrId(),addr.getLoc(),employe
e.getEmpId());

			}
			if(i1==1 && i==1){
				System.out.println("Data inserted successfully");
			}
		}
		public Address getAddressDetailsBasedOnEmpId(int empId){
			String sql="select a1.aid,a1.loc from emp e1 left outer
join addr a1 on e1.eid=a1.eid where e1.eid=?";
			Address addr=template.queryForObject(sql,new
RowMapper<Address>(){
				@Override
				publicAddress mapRow(ResultSet rs, int rowNum)
throws SQLException {
					Address addr=new Address();
					addr.setAddrId(rs.getInt(1));
					addr.setLoc(rs.getString(2));
					return addr;
				}
			},empId);
			return addr;
		}
}
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.app.Address;
importcom.app.Employee;
import com.app.EmployeeDaoImpl;


publicclass Test {

		public static void main(String[] args) {
			ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
			EmployeeDaoImpl daoImpl=context.getBean("empDao",
EmployeeDaoImpl.class);
			/* save record*/
			/* Employee employee=new Employee();
			employee.setEmpId(100);
			employee.setEmpName("ABCD");
			employee.setEmpSal(362.36);
			Address addr=new Address();
```

```
            addr.setAddrId(1236);
            addr.setLoc("HYD");
            //connect the address to employee
            employee.setAddr(addr);*/
            //daoImpl.insertDataToDB(employee);
            /*fetch emp address based on empid*/
            Address
address=daoImpl.getAddressDetailsBasedOnEmpId(100);
            System.out.println(address);

      }
}
```

If you get the output: try for delete and update…
All the best…

Spring JDBC-CURD Operations:

```
packagecom.app.model;

publicclass Employee {

      privateintempId;
      private String empName;
      privatedoubleempSal;

      public Employee() {
            super();
      }
      public Employee(intempId, String empName, doubleempSal) {
            super();
            this.empId = empId;
            this.empName = empName;
            this.empSal = empSal;
      }
      publicintgetEmpId() {
            returnempId;
      }
      publicvoidsetEmpId(intempId) {
            this.empId = empId;
      }
      public String getEmpName() {
            returnempName;
      }
      publicvoidsetEmpName(String empName) {
            this.empName = empName;
      }
      publicdoublegetEmpSal() {
```

```
		returnempSal;
	}
	publicvoidsetEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName
				+ ", empSal=" + empSal + "]";
	}
	@Override
	publicinthashCode() {
		finalint prime = 31;
		int result = 1;
		result = prime * result + empId;
		result = prime * result + ((empName == null) ? 0 :
empName.hashCode());
		long temp;
		temp = Double.doubleToLongBits(empSal);
		result = prime * result + (int) (temp ^ (temp >>> 32));
		return result;
	}
	@Override
	publicboolean equals(Object obj) {
		if (this == obj)
			returntrue;
		if (obj == null)
			returnfalse;
		if (getClass() != obj.getClass())
			returnfalse;
		Employee other = (Employee) obj;
		if (empId != other.empId)
			returnfalse;
		if (empName == null) {
			if (other.empName != null)
				returnfalse;
		} elseif (!empName.equals(other.empName))
			returnfalse;
		if (Double.doubleToLongBits(empSal) != Double
				.doubleToLongBits(other.empSal))
			returnfalse;
		returntrue;
	}

}
```

```
packagecom.app.dao;

importjava.util.List;
```

```
importcom.app.model.Employee;

publicinterfaceIEmployeeDao {

	publicintinsertEmployeeToDB(Employee employee);
	publicintupdateEmployeeById(Employee emp);
	publicintdeleteEmployeeById(intempId);
	public Employee getEmployeeObjectById(intempId);
	public List<Employee>getAllEmployessAsList();
}
```

```
packagecom.app.dao.impl;

importjava.sql.ResultSet;
importjava.sql.SQLException;
importjava.util.List;

importorg.springframework.beans.factory.annotation.Autowired;
importorg.springframework.jdbc.core.JdbcTemplate;
importorg.springframework.jdbc.core.RowMapper;
importorg.springframework.stereotype.Component;

importcom.app.dao.IEmployeeDao;
importcom.app.model.Employee;
@Component("empSpringJdbcDao")
publicclassEmployeeSpringJdbcDaoImplimplementsIEmployeeDao{

	@Autowired
	privateJdbcTemplatetemplate;

	@Override
	publicintinsertEmployeeToDB(Employee employee) {
		String sql="insert into empvalues(?,?,?)";
		returntemplate.update(sql,
employee.getEmpId(),employee.getEmpName(),employee.getEmpSal());
	}
	@Override
	publicintupdateEmployeeById(Employee emp) {
		String sql="update emp set ename=?,esal=? whereeid=?";
		returntemplate.update(sql,
emp.getEmpName(),emp.getEmpSal(),emp.getEmpId());
	}
	@Override
	publicintdeleteEmployeeById(intempId) {
		String sql="delete from emp where eid=?";
		returntemplate.update(sql, empId);
	}
	@Override
	public Employee getEmployeeObjectById(intempId) {
		String sql="select * from emp where eid=?";
```

```
            Employee emp=template.queryForObject(sql,
newRowMapper<Employee>(){
                  @Override
                  public Employee mapRow(ResultSetrs, introwNum)
                              throwsSQLException {
                        Employee emp=newEmployee();
                        emp.setEmpId(rs.getInt(1));
                        emp.setEmpName(rs.getString(2));
                        emp.setEmpSal(rs.getDouble(3));
                        returnemp;
                  }
            }, empId);
            returnemp;
      }
      @Override
      public List<Employee>getAllEmployessAsList() {
            String sql="select * from emp";
            List<Employee>empList=template.query(sql,
newRowMapper<Employee>(){
                  @Override
                  public Employee mapRow(ResultSetrs, introwNum)
                              throwsSQLException {
                        Employee emp=newEmployee();
                        emp.setEmpId(rs.getInt(1));
                        emp.setEmpName(rs.getString(2));
                        emp.setEmpSal(rs.getDouble(3));
                        returnemp;
                  }
            });
            returnempList;
      }
}
```

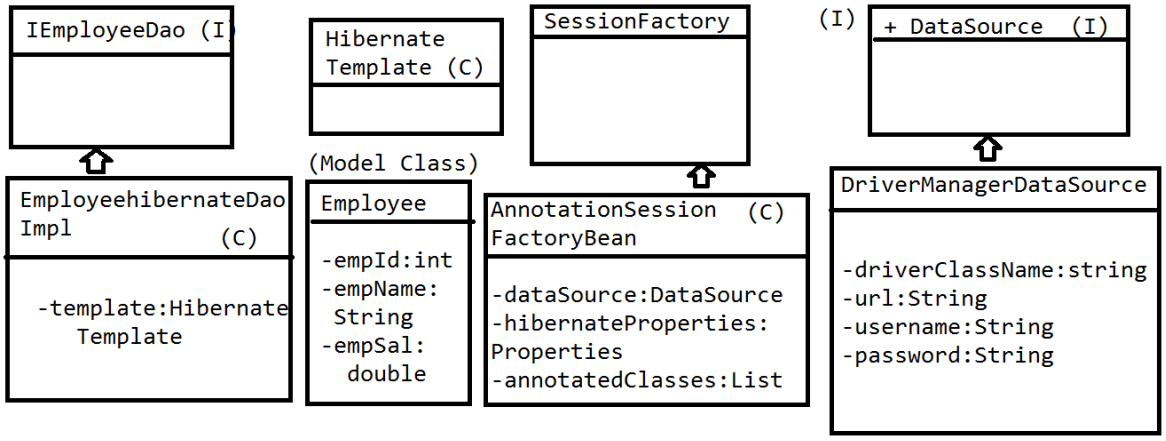
```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-
context.xsd">
      <context:annotation-config/>
      <context:component-scanbase-package="com.app"/>
      <context:property-placeholderlocation="jdbc.properties"/>
      <beanclass="org.springframework.jdbc.datasource.DriverManagerD
ataSource"name="dsObj">
            <propertyname="driverClassName"value="${driver}"/>
            <propertyname="url"value="${url}"/>
```

```
            <propertyname="username"value="system"/>
            <propertyname="password"value="system"/>
      </bean>
      <beanclass="org.springframework.jdbc.core.JdbcTemplate"name="t
empObj">
            <propertyname="dataSource">
                  <refbean="dsObj"/>
            </property>
      </bean>


</beans>
```

```
#JDBC Properties File
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=system
```

```
packagecom.app;

importjava.util.List;

importorg.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

importcom.app.dao.IEmployeeDao;
importcom.app.model.Employee;

publicclass Test {

      publicstaticvoid main(String[] args) {

      ApplicationContextcontext=newClassPathXmlApplicationContext("c
onfig.xml");

      IEmployeeDaoob=(IEmployeeDao)context.getBean("empSpringJdbcDao
");
            List<Employee>empList=ob.getAllEmployessAsList();
            System.out.println(empList);
      }
}
```

Spring-ORM(Hibernate) Example(Using Hibernate template):



config.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"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:util="http://www.springframework.org/schema/util"
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
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd
">

<context:annotation-config/>
<context:component-scan base-package="com.app"/>


     <bean
class="org.springframework.jdbc.datasource.DriverManagerDataSource"
name="dataSource">
          <property name="driverClassName"
value="com.mysql.jdbc.Driver"/>
          <property name="url"
value="jdbc:mysql://localhost:3306/test"/>
          <property name="username" value="root"/>
          <property name="password" value="root"/>

     </bean>

     <bean
class="org.springframework.orm.hibernate3.annotation.AnnotationSessi
onFactoryBean" name="sessionFactory">
          <property name="dataSource" ref="dataSource"/>
```

```
            <property name="hibernateProperties">
                  <props>
                        <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                        <prop key="hibernate.show_sql">true</prop>
                        <prop
key="hibernate.hbm2ddl.auto">update</prop>
                  </props>
            </property>
            <property name="annotatedClasses">
                  <list>
                        <value>com.app.Employee</value>
                  </list>
            </property>
      </bean>

      <bean
class="org.springframework.orm.hibernate3.HibernateTemplate"
name="template">
            <property name="sessionFactory"
ref="sessionFactory"></property>

      </bean>


      <bean class="com.app.EmployeeDaoHibernateImpl" name="empDao">
            <property name="template">
                  <ref bean="template"/>

            </property>

      </bean>
</beans>
```

Employee.java

```
package com.app;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="emp")
public class Employee {

      @Id
      @Column(name="eid")
```

```
	private Integer empId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	private Double empSal;

	public Employee() {
		super();
	}


	public Employee(Integer empId, String empName, Double empSal)
{
		super();
		this.empId = empId;
		this.empName = empName;
		this.empSal = empSal;
	}


	public Integer getEmpId() {
		return empId;
	}
	public void setEmpId(Integer empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public Double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(Double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" +
empName
				+ ", empSal=" + empSal + "]";
	}
}
```

```
package com.app;

import java.util.List;
```

```
public interface IEmployeeDao {
	public Integer createEmployee(Employee emp);
	public Integer updateEmployee(Employee emp);
	public Integer deleteEmployee(Integer empId);
	public Employee loadEmployee(Integer empId);
	public List<Employee> getAllEmployees();

}
```

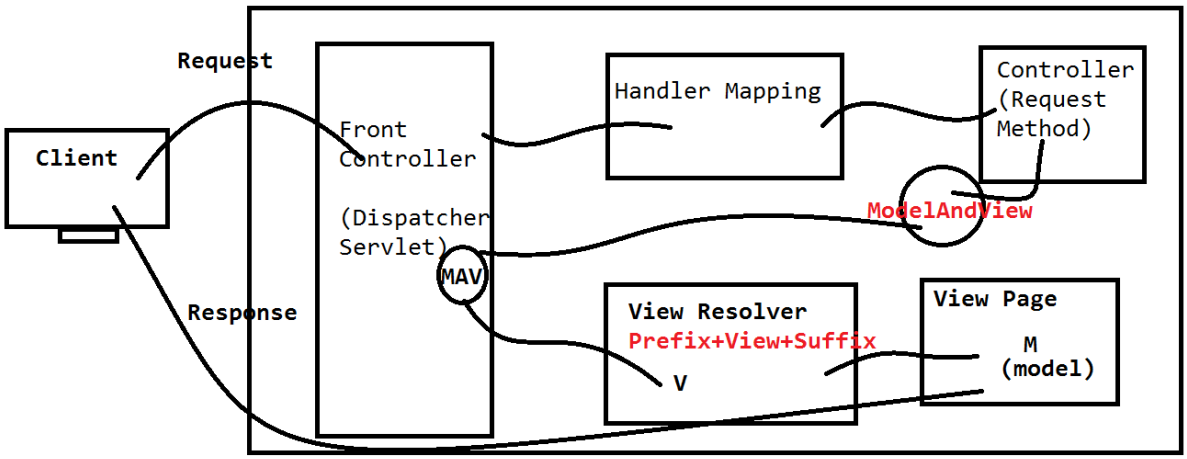
```
package com.app;

import java.util.List;

import org.springframework.orm.hibernate3.HibernateTemplate;

public class EmployeeDaoHibernateImpl implements IEmployeeDao {

	private HibernateTemplate template;


	public void setTemplate(HibernateTemplate template) {
		this.template = template;
	}

	@Override
	public Integer createEmployee(Employee emp) {
		Integer id=(Integer)template.save(emp);
		return id;
	}

	@Override
	public Integer updateEmployee(Employee emp) {
		template.update(emp);
		return emp.getEmpId();
	}


	@Override
	public Integer deleteEmployee(Integer empId) {
		Employee emp=template.get(Employee.class, empId);
		if (emp!=null) {
			template.delete(emp);
		}
		return emp.getEmpId();
	}


	@Override
```

```
	public Employee loadEmployee(Integer empId) {

		return template.get(Employee.class, empId);
	}


	@Override
	public List<Employee> getAllEmployees() {

		return template.loadAll(Employee.class);
	}



}
```

Main.java

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.app.Employee;
import com.app.IEmployeeDao;


public class Main {

	/**
	 * @param args
	 * @author Raghu
	 */
	public static void main(String[] args) {

		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		IEmployeeDao
empDao=(IEmployeeDao)context.getBean("empDao");
		Employee emp=new Employee(103,"ka", 16.5);
		System.out.println(empDao.getAllEmployees());
	}

}
```

Spring MVC Notes:



>Change Perspective to JavaEE
> Get a server view.
(Window->Showview->server)
>select a server , browse Head/Installed Location.
>Finish

File>new>Dynamic Web Project.
Provide Project name>next>next> Check web.xml generation
>Finish

Sample Proram:
JARS:
Copy and paste all Spring-JARS in lib folder.

Step 1:
Configure DispatcherServlet in web.xml

step 2: creating spring config file:
name rule: [<servlet-name>]-servlet.xml
Location: /WEB-INF/ location.

Step 3: Configure A view Resolver Class. With prefix and suffix.

Step 4: Controller
>Create a Java class.
>Use an annotation @Controller (Sterio type annot.)
>Enable annotations in config file
(use context schema for base pakage and annotation config)

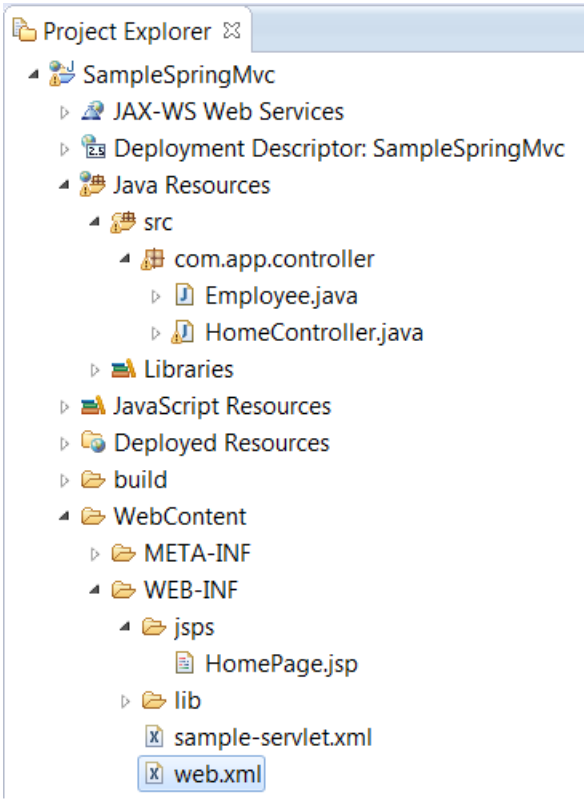
>Writre a request method. (@RequestMapping("/url"))

ex: @RequestMapping("/home") -- GET type
@RequestMapping(value="/abcd",method=RequestMethod.POST)

> Return a ModelAndView Object.
ex:
return new ModelAndView("HomePage");

Step 5: VIEW : create a folder structure for prefix
(if not exist ex: /WEB-INF/jsps)

Note: RequestMethod is enum, which define named constans
for request methods like GET,POST,PUT,TRACE...etc



Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
```

```
  <servlet>
      <servlet-name>sample</servlet-name>
      <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
  </servlet>
  <servlet-mapping>
      <servlet-name>sample</servlet-name>
      <url-pattern>/abcd/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Sample-servlet.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"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"

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

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

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd
                           ">
      <context:annotation-config/>
      <context:component-scan base-package="com.app.controller"/>

      <bean
class="org.springframework.web.servlet.view.InternalResourceViewReso
lver">
            <property name="prefix">
                  <value>/WEB-INF/jsps/</value>
            </property>
            <property name="suffix">
                  <value>.jsp</value>
            </property>

      </bean>



</beans>
```

HomeController.java

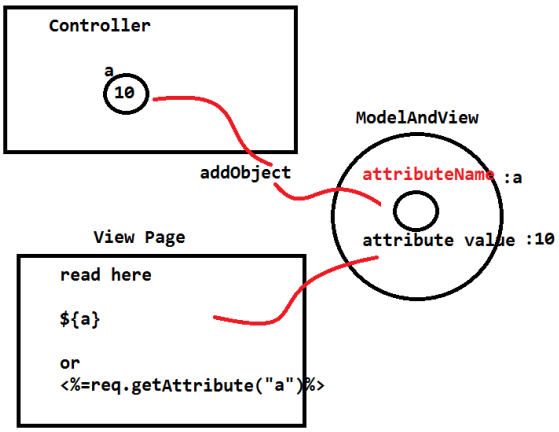
```java
package com.app.controller;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HomeController {

	@RequestMapping("/home")
	public ModelAndView showView(){

		ModelAndView mav=new ModelAndView();
		mav.setViewName("Home");
		return mav;
	}

	@RequestMapping(value="/home",method=RequestMethod.POST)
	public ModelAndView showView1(){

		return null;
	}


}
```

home.jsp

```
<h1>Welcome to Spring</h1>
```

URL:- http://localhost:8080/SampleSpringMVC/abcd/home

Spring Sending Data From Controller to UI:-



ModelAndView is a common Object to send data controller and UI page. ModelAndView had a method addObject(String key,Object value), is used to add data to MAV Object. Using this same key we have to retrieve data at UI level.

Ex:

Sending Primitive Values at Controller:

```
packagecom.app;

importorg.springframework.stereotype.Controller;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestParam;
importorg.springframework.web.bind.annotation.ResponseBody;
importorg.springframework.web.servlet.ModelAndView;

@Controller
publicclassHomeController {

	@RequestMapping("/welcome")
	publicModelAndView welcome(){
		ModelAndViewmav=newModelAndView("welcome");
		mav.addObject("empId", 100);
		mav.addObject("empName", "AJ");
		mav.addObject("empSal", 250.36);
		returnmav;
	}

}
```

JSP Code to Read Data:

Welcome.jsp

```
<h1>Data is :</h1>

<b>${empId},${empName},${empSal}</b>

<%=req.getAttribute(“empId”)%>
```

Sending Object Data at Controller:

```
packagecom.app;

importorg.springframework.stereotype.Controller;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestParam;
importorg.springframework.web.bind.annotation.ResponseBody;
importorg.springframework.web.servlet.ModelAndView;

@Controller
publicclassHomeController {

	@RequestMapping("/welcome")
	publicModelAndView welcome(){
		ModelAndViewmav=newModelAndView("welcome");
		Employeeemp=newEmployee();
		emp.setEmpId(10);
		emp.setEmpName("AJ");
		emp.setEmpSal(12.35);
		mav.addObject("empObj", emp);
		returnmav;
	}

}
```

Reading data at JSP page:-

```
<b> ${empObj.empId},${empObj.empName},${empObj.empSal}</b>
```

Or

```
<% Employee emp=req.getAttribute(“empObj”);

Out.print(emp.getEmpId());

%>
```

Sending Collection Data from Controller:

```
packagecom.app;
```

```
importorg.springframework.stereotype.Controller;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestParam;
importorg.springframework.web.bind.annotation.ResponseBody;
importorg.springframework.web.servlet.ModelAndView;

@Controller
publicclassHomeController {

	@RequestMapping("/welcome")
	publicModelAndView welcome(){
		ModelAndViewmav=newModelAndView("welcome");
		Employeeemp=newEmployee();
		emp.setEmpId(10);
		emp.setEmpName("AJ");
		emp.setEmpSal(12.35);

		Employee emp2=newEmployee();
		emp2.setEmpId(10);
		emp2.setEmpName("AJ");
		emp2.setEmpSal(12.35);

		Employee emp3=newEmployee();
		emp3.setEmpId(10);
		emp3.setEmpName("AJ");
		emp3.setEmpSal(12.35);

		List<Employee>empListObj=newArrayList<Employee>();
		empListObj.add(emp);
		empListObj.add(emp2);
		empListObj.add(emp3);
		mav.addObject("empListObj", empListObj);
		returnmav;
	}

}
```

Reading Data At JSP Level:

```
<c:forEachitems="${empListObj}"var="emp">
<c:outvalue="${emp.empId}"/>,<c:outvalue="${emp.empName}"/>,<c:outva
lue="${emp.empSal}"/><br/>
</c:forEach>

or

<%
ListempList=request.getAttribute("empListObj");
Iterator iterator= empList.iterator();
```

```
while(iterator.hasNext()){
      Employeeemp=(Employee)iterator.next();
      out.println(emp);
}
%>
```

For JSTL code:

Add below line in JSP:

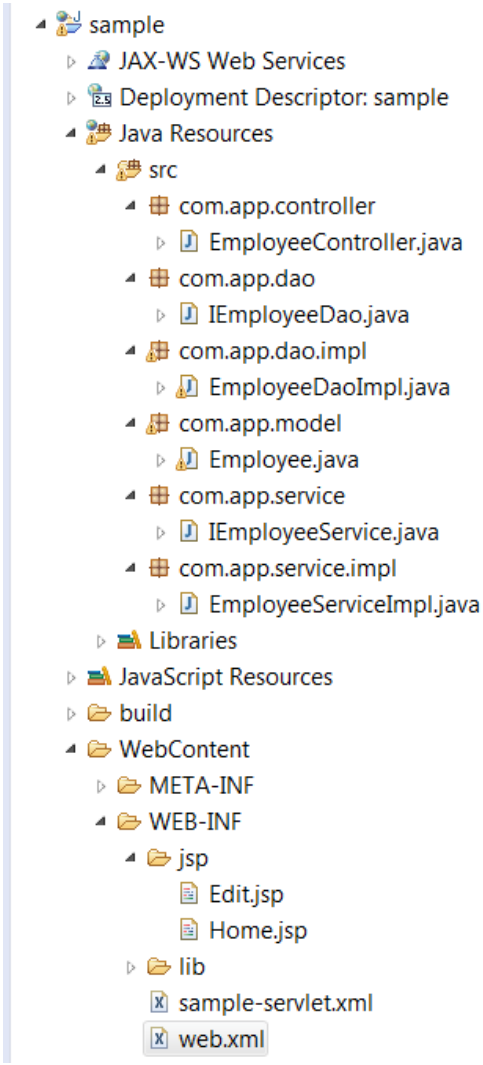
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

Download and Add JSTL JAR in lib folder:

Location :http://central.maven.org/maven2/javax/servlet/jstl/1.2/jstl-1.2.jar

Spring MVC (Sending Data From Controller to UI).
JSTL JAR Location :
http://central.maven.org/…/ja…/servlet/jstl/1.2/jstl-1.2.jar
(Download Link). Add this in lib folder.

Spring –Hibernate integration:



Web.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<web-appxmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xmlns="http://java.sun.com/xml/ns/javaee"xmlns:web="http://
java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"id="WebApp_ID"version="2.5">
<display-name>sample</display-name>

<servlet>
      <servlet-name>sample</servlet-name>
      <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
      <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
```

```
	<servlet-name>sample</servlet-name>
	<url-pattern>/app/*</url-pattern>
</servlet-mapping>
</web-app>
```

Sample-servlet.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:util="http://www.springframework.org/schema/util"
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
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd
">

	<context:annotation-config/>
	<context:component-scanbase-package="com.app"/>

	<beanclass="org.springframework.web.servlet.view.InternalResou
rceViewResolver">
		<propertyname="prefix"value="/WEB-INF/jsp/"/>
<propertyname="suffix"value=".jsp"/>
	</bean>

	<beanclass="org.springframework.jdbc.datasource.DriverManagerD
ataSource"name="datasource">

	<propertyname="driverClassName"value="com.mysql.jdbc.Driver"/>

	<propertyname="url"value="jdbc:mysql://localhost:3306/test"/>
		<propertyname="username"value="root"/>
		<propertyname="password"value="root"/>
	</bean>

	<beanclass="org.springframework.orm.hibernate3.annotation.Anno
tationSessionFactoryBean"name="sessionFactory">
		<propertyname="dataSource">
			<refbean="datasource"/>
		</property>
		<propertyname="annotatedClasses">
			<list>
				<value>com.app.model.Employee</value>
			</list>
		</property>
```

```
            <propertyname="hibernateProperties">
                  <props>

      <propkey="hibernate.dialect">org.hibernate.dialect.MySQLDialec
t</prop>
                        <propkey="hibernate.show_sql">true</prop>

      <propkey="hibernate.hbm2ddl.auto">update</prop>
                  </props>
            </property>
      </bean>


</beans>
```

EmployeeController.java

```
package com.app.controller;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.app.model.Employee;
import com.app.service.IEmployeeService;


@Controller
publicclass EmployeeController {

      @Autowired
      private IEmployeeService empService;

      @RequestMapping("/home")
      public ModelAndView homePage(){

            returnnew ModelAndView("Home");
      }

      @RequestMapping("/insert")
      public String insert(@ModelAttribute("employee")Employee emp){
            empService.createEmployee(emp);
            System.out.println("hello");
            return"redirect:/app/show";
```

```
	}

	@RequestMapping("/show")
	public ModelAndView show(@ModelAttribute("employee")Employee
emp){
		List<Employee>
employeeList=empService.showEmployeeDetails();
		returnnew
ModelAndView("Home","employeeList",employeeList);

	}
	@RequestMapping("/delete/{empId}")
	public String delete(@PathVariable("empId")int empId){
		empService.deleteEmployeeById(empId);
		return"redirect:/app/show";
	}

	@RequestMapping("/edit/{empId}")
	public ModelAndView edit(@PathVariable("empId")int empId){
		Employee emp=empService.loadEmployeeById(empId);
		returnnew ModelAndView("Edit","emp",emp);
	}
}
```

IEmployeeService.java

```
package com.app.service;

import java.util.List;

importcom.app.model.Employee;

publicinterface IEmployeeService {

	publicvoid createEmployee(Employee employee);
	public List<Employee> showEmployeeDetails();
	publicvoid deleteEmployeeById(int empId);
	publicEmployee loadEmployeeById(int empId);
}
```

EmployeeServiceImpl.java

```
package com.app.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import com.app.dao.IEmployeeDao;
import com.app.model.Employee;
import com.app.service.IEmployeeService;

@Service("empService")
publicclass EmployeeServiceImpl implements IEmployeeService {

	@Autowired
	private IEmployeeDao empDao;

	@Override
	publicvoid createEmployee(Employee employee) {
		try {
			empDao.createEmployee(employee);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	@Override
	public List<Employee> showEmployeeDetails() {
	List<Employee> employeeList=null;
		try {
			employeeList=empDao.showEmployeeDetails();
		} catch (Exception e) {
			// TODO: handle exception
		}
		return employeeList;
	}
	@Override
	publicvoid deleteEmployeeById(int empId) {
		try {
			empDao.deleteEmployeeById(empId);
		} catch (Exception e) {
			e.printStackTrace();
		}

	}
	@Override
	public Employee loadEmployeeById(int empId) {
		Employee emp=null;
		try {
			emp=empDao.loadEmployeeById(empId);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return emp;
	}

}
```

IEmployeeDao.java

```
package com.app.dao;

import java.util.List;

importcom.app.model.Employee;

publicinterface IEmployeeDao {

	publicvoid createEmployee(Employee employee);
	public List<Employee> showEmployeeDetails();
	publicvoid deleteEmployeeById(int empId);
	publicEmployee loadEmployeeById(int empId);
}
```

EmployeeDaoImpl.java

```
package com.app.dao.impl;


import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.app.dao.IEmployeeDao;
import com.app.model.Employee;

@Repository("empDao")
publicclass EmployeeDaoImpl implements IEmployeeDao{

	@Autowired
	private SessionFactory sessionFactory;

	@Override
	publicvoid createEmployee(Employee employee) {
		Session session=null;
		Transaction tx=null;
		try {

			session=sessionFactory.openSession();
```

```
			tx=session.beginTransaction();
			tx.begin();
			session.saveOrUpdate(employee);
			tx.commit();
		} catch (Exception e) {
			tx.rollback();
			e.printStackTrace();

		}finally{
			session.flush();
			session.close();
		}
	}

	@Override
	public List<Employee> showEmployeeDetails() {
		Session session=null;
		Transaction tx=null;
		Criteria criteria=null;
		List<Employee> employeeList=null;
		try {

			session=sessionFactory.openSession();
			tx=session.beginTransaction();
			tx.begin();
			criteria=session.createCriteria(Employee.class);
			employeeList=criteria.list();
			tx.commit();
		} catch (Exception e) {
			tx.rollback();
			e.printStackTrace();

		}finally{
			session.flush();
			session.close();
		}

		return employeeList;
	}


	@Override
	publicvoid deleteEmployeeById(int empId) {
		Session session=null;
		Transaction tx=null;
		Employee emp=null;
		try {

			session=sessionFactory.openSession();
```

```
			tx=session.beginTransaction();
			tx.begin();
			emp=(Employee)session.get(Employee.class, empId);
			session.delete(emp);
			tx.commit();
		} catch (Exception e) {
			tx.rollback();
			e.printStackTrace();

		}finally{
			session.flush();
			session.close();
		}

	}
	@Override
	public Employee loadEmployeeById(int empId) {
		Session session=null;
		Employee emp=null;
		try {

			session=sessionFactory.openSession();
			emp=(Employee)session.get(Employee.class, empId);
		} catch (Exception e) {
			e.printStackTrace();

		}finally{
			session.flush();
			session.close();
		}
		return emp;
	}
}
```

Home.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@taglibprefix="c"uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html; charset=ISO-8859-
1">
<title>Insert title here</title>
<styletype="text/css">
tabletrth{
	color: white;
	background-color: black;
```

```
}
</style>
</head>
<body>
	<h1>Wel come page</h1>


<formaction="insert"method="POST">
<pre>

user Id:
<inputtype="number"name="empId"value="${emp.empId}"required="require
d"min="10"max="200">
user name:
<inputtype="text"name="empName"value="${emp.empName}"required="requi
red">
user sal :
<inputtype="text"name="empSal"value="${emp.empSal}"required="require
d">
<inputtype="submit"value="Insert">
</pre>
</form>
<c:iftest="${!empty employeeList}">
<tableborder="1">
	<tr>
		<th>EMP ID</th><th>EMP NAME</th><th>EMP SAL</th>
	</tr>
	<c:forEachitems="${employeeList}"var="employee">
		<tr>

	<td><ahref="edit/${employee.empId}"><c:outvalue="${employee.em
pId}"/></td>
			<td><c:outvalue="${employee.empName}"/></td>
			<td><c:outvalue="${employee.empSal}"/></td>

		</tr>
	</c:forEach>
</table>
</c:if>
</body>
</html>
```

Edit.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<metahttp-equiv="Content-Type"content="text/html; charset=ISO-8859-
1">
<title>Insert title here</title>
</head>
<body>
<formaction="../insert"method="POST">
<pre>

user Id:
<inputtype="number"name="empId"value="${emp.empId}"required="require
d"min="10"max="200">
user name:
<inputtype="text"name="empName"value="${emp.empName}"required="requi
red">
user sal :
<inputtype="text"name="empSal"value="${emp.empSal}"required="require
d">
<inputtype="submit"value="Insert">
</pre>
</form>
</body>
</html>
```

AJAX: Example Program:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<head>
<script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
<script type="text/javascript">
function makeCall(){
      var input1=$("#abcd").val();
      $.ajax({
            url:'ajaxCall',
            data:'input='+input1,
            success:function(message){
                  $("#myDivArea").html(message);
            }

      });

}

</script>


<title>Insert title here</title>
```

```
</head>
<body>
<div id="myDivArea"></div>


<input type="button" value="Show Out" onclick="return makeCall();">
<input type="text" name="abcd" id="abcd">
</body>
</html>
```

```
package com.app;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HomeController {

	@RequestMapping("/welcome")
	public ModelAndView showPage(){

		return new ModelAndView("welcome");

	}

	@RequestMapping("/ajaxCall")
	public @ResponseBody String
provideResponse(@RequestParam("input")String input){

		return input+" "+Math.random()*10000;
	}


}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <servlet>
	<servlet-name>sample</servlet-name>
```

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

  <servlet-mapping>
      <servlet-name>sample</servlet-name>
      <url-pattern>/app/*</url-pattern>
  </servlet-mapping>
</web-app>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
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/>
   <context:component-scan base-package="com.app"/>

      <bean
class="org.springframework.web.servlet.view.InternalResourceViewReso
lver">
            <property name="prefix" value="/WEB-INF/jsps/"/>
            <property name="suffix" value=".jsp"/>
      </bean>




</beans>
```

Delete Employee Using Path variable:

```
package com.app;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HomeController {

	@Autowired
	@Qualifier("empService")
	private IEmpService empService;


	@RequestMapping(value="/register")
	public ModelAndView insertRecord(){
		return new ModelAndView("Register");
	}


	@RequestMapping(value="/home/{empId}",method=RequestMethod.GET
)
	public ModelAndView sayHello(@PathVariable("empId")String
empid){
		System.out.println("Entered vales is"+empid);
		return new ModelAndView("Page1");
	}

	@RequestMapping(value="/insert")
	public ModelAndView
insertRecord(@ModelAttribute("employee")Employee emp){
		empService.insertEmployee(emp);
		List<Employee> empList=empService.getAllEmployee();

		return new ModelAndView("Register","empList",empList);
	}

	@RequestMapping("/delete/{empId}")
	public ModelAndView deleteRecord(@PathVariable("empId")Integer
empId){
		empService.deleteEmployee(empId);
		List<Employee> empList=empService.getAllEmployee();

		return new ModelAndView("Register","empList",empList);
	}



}
```

```
package com.app;

import java.util.List;


public interface IEmpService {

	public Integer insertEmployee(Employee emp);
	public List<Employee> getAllEmployee();
	public void deleteEmployee(Integer empId);
}

----------]
package com.app;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service("empService")
public class EmpServiceImpl implements IEmpService {

	@Autowired
	@Qualifier("empDao")
	private IEmpDao empDao;

	@Override
	public Integer insertEmployee(Employee emp) {
		return empDao.insertEmployee(emp);
	}

	@Override
	public List<Employee> getAllEmployee() {
		return empDao.getAllEmployee();
	}

	@Override
	public void deleteEmployee(Integer empId) {
		empDao.deleteEmployee(empId);

	}

}
```

```
package com.app;

import java.util.List;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service("empService")
public class EmpServiceImpl implements IEmpService {

	@Autowired
	@Qualifier("empDao")
	private IEmpDao empDao;

	@Override
	public Integer insertEmployee(Employee emp) {
		return empDao.insertEmployee(emp);
	}

	@Override
	public List<Employee> getAllEmployee() {
		return empDao.getAllEmployee();
	}

	@Override
	public void deleteEmployee(Integer empId) {
		empDao.deleteEmployee(empId);

	}

}
```

```java
package com.app;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="emp")
public class Employee {

	@Id
	@Column(name="eid")
	private Integer empId;
	@Column(name="ename")
	private String empName;


	public Integer getEmpId() {
```

```
		return empId;
	}
	public void setEmpId(Integer empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}


}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="insert">
	<pre>
	User Id :<input type="text" name="empId" id="empId">
   User Name: <input type="text" name="empName" id="empName">
   <input type="submit" value="Register">
   </pre>
</form>

<c:forEach items="${empList}" var="emp">
	<c:out value="${emp.empId}"/>
	<c:out value="${emp.empName}"/>
	<a href='delete/<c:out value="${emp.empId}"/>'/>Delete</a>



	<br/>
</c:forEach>

</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
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/>
   <context:component-scan base-package="com.app"/>

      <bean
class="org.springframework.web.servlet.view.InternalResourceViewReso
lver">
            <property name="prefix" value="/WEB-INF/jsps/"/>
            <property name="suffix" value=".jsp"/>
      </bean>

      <bean name="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
         <property name="driverClassName"
value="com.mysql.jdbc.Driver" />
         <property name="url"
value="jdbc:mysql://localhost:3306/test" />
         <property name="username" value="root" />
         <property name="password" value="root" />
    </bean>

    <bean name="mySessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessi
onFactoryBean">
         <property name="dataSource" ref="dataSource"/>
            <property name="annotatedClasses">
             <list>
                        <value>com.app.Employee</value>
            </list>
            </property>
         <property name="hibernateProperties">
             <props>
                        <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                        <prop
key="hibernate.hbm2ddl.auto">update</prop>
                        <prop key="hibernate.show_sql">true</prop>
                  </props>
         </property>
    </bean>
```

```
    <bean
class="org.springframework.orm.hibernate3.HibernateTemplate"
name="template">
      <property name="sessionFactory" ref="mySessionFactory"/>

    </bean>




</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">



  <servlet>
      <servlet-name>sample</servlet-name>
      <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
  </servlet>

  <servlet-mapping>
      <servlet-name>sample</servlet-name>
      <url-pattern>/app/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Aspect Oriented Programming :- Without changing business logic, a new service can be added to existed layer, using Aspect and advices we can construct services.

Aspect : it is service class.

Advice: It represents a business method

Pointcut: It is expression, it will select business methods to be bounded with advices

Aspect:

```
package com.app;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class SampleAspectEx {

	@Before("execution(* com.one.*.*.get*(..))")
	public void showMessage(){

		System.out.println("Hello I'm from aspect");
	}
}
```

Circle:

```
package com.one.two;

public class Circle {

	private String name;
	private int type;

	public String getName() {
		System.out.println("In-Circle-In-getName():String");
		return name;
	}

	public String getName(int id) {
		System.out.println("In-Circle-In-getName(int):String");
		return name;
	}

	public void setName(String name) {
```

```
		this.name = name;
	}



	public int getType() {
		System.out.println("In-Circle-In-getType():int");
		return type;
	}

	public int getType(int id) {
		System.out.println("In-Circle-In-getType(int):int");
		return type;
	}


	public void setType(int type) {
		this.type = type;
	}




}
```

Shape:

```
package com.one.three;

public class Shape {

	private String name;
	private int type;

	public String getName() {
		System.out.println("In-Shape-In-getName():String");
		return name;
	}

	public String getName(int id) {
		System.out.println("In-Shape-In-getName(int):String");
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}
```

```
	public int getType() {
		System.out.println("In-Shape-In-getType():int");
		return type;
	}

	public int getType(int id) {
		System.out.println("In-Shape-In-getType(int):int");
		return type;
	}


	public void setType(int type) {
		this.type = type;
	}




}
```

Student:

```
package com.app.module;

public class Student {

	private String name;
	private int type;

	public String getName() {
		System.out.println("In-Student-In-getName():String");
		return name;
	}

	public String getName(int id) {
		System.out.println("In-Student-In-getName(int):String");
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}
```

```
	public int getType() {
		System.out.println("In-Student-In-getType():int");
		return type;
	}

	public int getType(int id) {
		System.out.println("In-Student-In-getType(int):int");
		return type;
	}


	public void setType(int type) {
		this.type = type;
	}


}
```

Rectangle:

```
package com.one;

public class Rectangle {

	private String name;
	private int type;

	public String getName() {
		System.out.println("In-Rectangle-In-getName():String");
		return name;
	}

	public String getName(int id) {
		System.out.println("In-Rectangle-In-
getName(int):String");
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}



	public int getType() {
		System.out.println("In-Rectangle-In-getType():int");
		return type;
	}

	public int getType(int id) {
```

```
		System.out.println("In-Rectangle-In-getType(int):int");
		return type;
	}


	public void setType(int type) {
		this.type = type;
	}



}
```

Employee:

```
package com.app;

public class Employee {

	private String name;
	private int type;

	public String getName() {
		System.out.println("In-Employee-In-getName():String");
		return name;
	}

	public String getName(int id) {
		System.out.println("In-Employee-In-getName(int):String");
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}



	public int getType() {
		System.out.println("In-Employee-In-getType():int");
		return type;
	}

	public int getType(int id) {
		System.out.println("In-Employee-In-getType(int):int");
		return type;
	}
```

```
	public void setType(int type) {
		this.type = type;
	}

}
```

Design:

```
package com.one.two;


public class Design {


	private String name;
	private int type;

	public String getName() {
		System.out.println("In-Design-In-getName():String");
		return name;
	}

	public String getName(int id) {
		System.out.println("In-Design-In-getName(int):String");
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getType() {
		System.out.println("In-Design-In-getType():int");
		return type;
	}

	public int getType(int id) {
		System.out.println("In-Design-In-getType(int):int");
		return type;
	}
	public void setType(int type) {
		this.type = type;
	}



}
```

Config.xml:

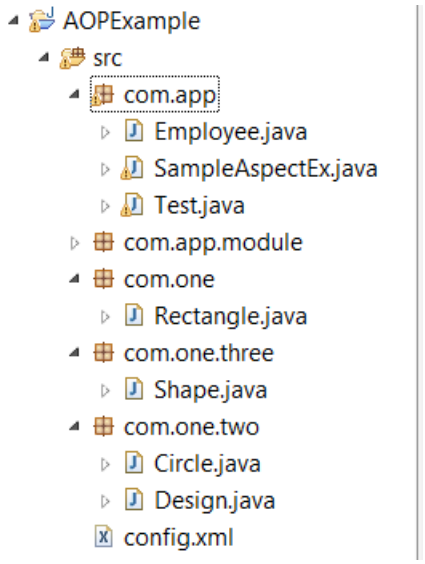
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"

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

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

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/aop

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

                <aop:aspectj-autoproxy/>

                <bean class="com.app.Employee" name="emp">
                     <property name="name">
                          <value>employee</value>
                     </property>
                     <property name="type">
                          <value>10</value>
                     </property>

                </bean>
                <bean class="com.one.two.Circle" name="c1">
                     <property name="name">
                          <value>circle</value>
                     </property>
                     <property name="type">
                          <value>20</value>
                     </property>

                </bean>
                <bean class="com.one.Rectangle" name="r1">
                     <property name="name">
                          <value>rectangle</value>
                     </property>
                     <property name="type">
                          <value>30</value>
                     </property>
```

```
		</bean>

		<bean class="com.one.three.Shape" name="s1">
			<property name="name">
				<value>shape</value>
			</property>
			<property name="type">
				<value>40</value>
			</property>

		</bean>

		<bean class="com.app.module.Student" name="std">
			<property name="name">
				<value>student</value>
			</property>
			<property name="type">
				<value>50</value>
			</property>

		</bean>

		 <bean class="com.one.two.Design" name="d1">
			<property name="name">
				<value>design</value>
			</property>
			<property name="type">
				<value>60</value>
			</property>

		</bean>

	<bean class="com.app.SampleAspectEx"></bean>


</beans>
```

Test class:

```
package com.app;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.app.module.Student;
import com.one.Rectangle;
import com.one.three.Shape;
```

```
import com.one.two.Circle;
import com.one.two.Design;

public class Test {

	public static void main(String[] args) {

		ApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
		Employee e=context.getBean("emp",Employee.class);
		Student std=context.getBean("std",Student.class);
		Shape s1=context.getBean("s1",Shape.class);
		Circle c1=context.getBean("c1",Circle.class);
		Rectangle r1=context.getBean("r1",Rectangle.class);
		Design d1=context.getBean("d1",Design.class);

		//(* com.one.Rectangle.get*())


		c1.getName(10);
		d1.getName(20);
		d1.getName();
		c1.getName();


		r1.getName();
		r1.getType();
		r1.getName(10);
		r1.getType(10);

	}

}
```



JARS:

AOP JARS(only):

http://www.mediafire.com/download/2szvk5sjxgoqkdp/Maven.rar

Also include cglib jar.

http://central.maven.org/maven2/cglib/cglib/3.1/cglib-3.1.jar

What are the benefits of using Spring Framework?

Following is the list of few of the great benefits of using Spring Framework --

- With the Dependency Injection(DI) approach, dependencies are explicit and evident in constructor or JavaBean properties.
- IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
- Spring does not reinvent the wheel instead, it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.

- Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.
- Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.
- Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.
- 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).

What is Inversion of Control (IoC) and Dependency Injection?

Inversion of control (IoC) is a programming technique in which object coupling is bound at run time. **Inversion of control is a design paradigm with the goal of giving more control to the targeted components of your application, the ones that are actually doing the work.**

Dependency injection is a pattern used to create instances of objects that other objects rely on without knowing at compile time which class will be used to provide that functionality. Inversion of control relies on dependency injection because a mechanism is needed in order to activate the components providing the specific functionality.

In Java, dependency injection may happen through 3 ways:

A constructor injection
A setter injection
An interface injection (Not supported by Spring)

How to inject a java.util.Properties into a Spring Bean?

First way is to use <props> tag as below.

```
<bean id="adminUser" class="com.app.common.Customer">
      <!-- java.util.Properties -->
    <property name="emails">
        <props>
            <prop key="admin">admin@nospam.com</prop>
            <prop key="support">support@nospam.com</prop>
        </props>
```

```
    </property>

</bean>
```

You can use “util:” namespace as well to create properties bean from properties file, and use bean reference for setter injection.

```
<util:properties id="emails" location="com/app/emails.properties" />
```

How do you turn on annotation based autowiring?

To enable @Autowired, you have to register AutowiredAnnotationBeanPostProcessor, and you can do it in two ways.

1. Include <context:annotation-config> in bean configuration file.

```
<beans>
    <context:annotation-config />
</beans>
```

2. Include AutowiredAnnotationBeanPostProcessor directly in bean configuration file.

```
<beans>
  <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
</beans>
```

Explain @Required annotation with example?

we can set “dependency-check” attribute of <bean> and set one of four attributes i.e. none, simple, objects or all (none is default option).

In case one application, we will not be interested in checking all the bean properties configured in your context files. Rather you would like to check if particular set of properties have been set or not in some specific beans only. Spring’s dependency checking feature using “dependency-check” attribute, will not able to help you in this case. So solve this problem, you can use @Required annotation.

To Use the @Required annotation over setter method of bean property in class file as below:

```
public class EmployeeFactoryBean {
@Required
private String designation;

    public String getDesignation() {
        return designation;
```

```
    }

        public void setDesignation(String designation) {
        this.designation = designation;
    }

    //more code here
}
```

Explain @Autowired annotation with example?

The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished.
The @Autowired annotation can be used to autowire bean on the setter method just like@Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

E.g. You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods, it tries to performbyType autowiring on the method.
You can apply @Autowired to constructors as well. A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean, even if no <constructor-arg> elements are used while configuring the bean in XML file.

```
public class TextEditor {
   private SpellCheckerspellChecker;

   @Autowired
   public TextEditor(SpellCheckerspellChecker){
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }

   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

And it’s configuration without constructor arguments.

```
<beans>

   <context:annotation-config/>
```

```
   <!-- Definition for textEditor bean without constructor-arg  -->
   <bean id="textEditor" class="com.howtodoinjava.TextEditor">
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.howtodoinjava.SpellChecker">
   </bean>

</beans>
```

Explain @Qualifier annotation with example?

@Qualifier means, which bean is qualify to autowired on a field. The qualifier annotation helps disambiguate bean references when Spring would otherwise not be able to do so.

See below example, it willautowired a “person” bean into customer’s person property.

```
public class Customer
{
    @Autowired
    private Person person;
}
```

And we have two bean definitions for Person class.

```
<bean id="customer" class="com.howtodoinjava.common.Customer" />

<bean id="personA" class="com.howtodoinjava.common.Person" >
    <property name="name" value="lokesh" />
</bean>

<bean id="personB" class="com.howtodoinjava.common.Person" >
    <property name="name" value="alex" />
</bean>
```

Will Spring know which person bean should autowired? NO. When you run above example, it hits below exception :

```
Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException:
    No unique bean of type [com.howtodoinjava.common.Person] is defined:
        expected single matching bean but found 2: [personA, personB]
```

To fix above problem, you need @Quanlifier to tell Spring about which bean should autowired.

```
public class Customer
{
    @Autowired
    @Qualifier("personA")
    private Person person;
```

}

Difference between constructor injection and setter injection?

In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives. In constructor injection, partial injection of dependencies is not possible, because for calling constructor we must pass all the arguments right, if not so we may get error.

Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property. But, constructor injection cannot overrides the setter injected values. It's obvious because constructors are called to first to create the instance.
Using setter injection you can not guarantee that certain dependency is injected or not, which means you may have an object with incomplete dependency. On other hand constructor Injection does not allow you to construct object, until your dependencies are ready.

In constructor injection, if Object A and B are dependent each other i.e A is depends on B and vice-versa, Spring throws ObjectCurrentlyInCreationException while creating objects of A and B because A object cannot be created until B is created and vice-versa. So spring can resolve circular dependencies through setter-injection because Objects are constructed before setter methods invoked.

How do you define the scope of a bean?

When defining a <bean> in Spring, we can also declare a scope for the bean. It can be defined through the scope attribute in the bean definition. For example, when Spring has to produce a new bean instance each time one is needed, the bean'sscope attribute to be prototype. On the other hand, when the same instance of a bean must be returned by Spring every time it is needed, the the bean scope attribute must be set to singleton.

Explain the bean scopes supported by Spring

There are five scoped provided by the Spring Framework supports following five scopes:

In singleton scope, Spring scopes the bean definition to a single instance per Spring IoC container.
In prototype scope, a single bean definition has any number of object instances.
In request scope, a bean is defined to an HTTP request. This scope is valid only in a web-aware Spring ApplicationContext.
In session scope, a bean definition is scoped to an HTTP session. This scope is also valid only in a web-aware Spring ApplicationContext.
In global-session scope, a bean definition is scoped to a global HTTP session. This is also a case used in a web-aware Spring ApplicationContext.
The default scope of a Spring Bean is Singleton.

What are inner beans in Spring?
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.

Explain different modes of auto wiring?
The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

no: This is default setting. Explicit bean reference should be used for wiring.

byName: When autowiring byName, the Spring container looks at the properties of the beans on which autowireattribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.

byType: When autowiring by datatype, the Spring container looks at the properties of the beans on which autowireattribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.

constructor: This mode is similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
autodetect: Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowirebybyType.

Design Patterns Used in Java Spring Framework

Dependency injection/ or IoC (inversion of control) – Is the main principle behind decoupling process that Spring does

Factory – Spring uses factory pattern to create objects of beans using Application Context reference

```
// Spring uses factory pattern to create instances of the objects
BeanFactory factory
= new XmlBeanFactory(new FileSystemResource("spring.xml"));
Triangle triangle = (Triangle) factory.getBean("triangle");
triangle.draw();
```

Proxy – used heavily in AOP, and remoting.

Singleton – by default, beans defined in spring config file (xml) are only created once. No matter how many calls were made using getBean() method, it will always have only one bean. This is because, by default all beans in spring are singletons.
This can be overridden by using Prototype bean scope.Then spring will create a new bean object for every request.

Model View Controller – The advantage with Spring MVC is that your controllers are POJOs as opposed to being servlets. This makes for easier testing of controllers. One thing to note is that the controller is only required to return a logical view name, and the view selection is left to a separate ViewResolver. This makes it easier to reuse controllers for different view technologies.

Front Controller – Spring provides DispatcherServlet to ensure an incoming request gets dispatched to your controllers.

View Helper – Spring has a number of custom JSP tags, and velocity macros, to assist in separating code from presentation in views.

Template method – used extensively to deal with boilerplate repeated code (such as closing connections cleanly, etc..). For example JdbcTemplate, JmsTemplate, JpaTemplate.

What is Aspect, Advice, Pointcut, JointPoint in AOP?

Aspect: Aspect is a class that implements cross-cutting concerns, such as logger, encryption.

Aspects can be a normal class configured and then configured in Spring Bean configuration file or we can use Spring AspectJ support to declare a class as Aspect using @Aspect annotation.

Advice: Advice is the action taken for a particular join point. In terms of programming, they are methods that gets executed when a specific join point with matching pointcut is reached in the application. You can think of Advices as Spring interceptors or Servlet Filters.

Pointcut: Pointcut are regular expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points. Spring framework uses the AspectJ pointcut expression language for determining the join points where advice methods will be applied.

Join Point: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

@Component is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used.

@Controller is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation. Handles mainly servlet request and response

@Repository 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. Handles DB Connection and

@Service is used to indicate that a class is a Service. Usually the business facade classes that provide some services are annotated with this.

We can use any of the above annotations for a class for auto-detection but different types are provided so that we can avoid conversion of objects as per layers.

How to achieve localization/Multilanguage in Spring MVC applications?

- Spring provides excellent support for localization or i18n through resource bundles. Basis steps needed to make our application localized are:
- Creating message resource bundles for different locales, such as messages_en.properties, messages_fr.properties etc.
- Defining messageSource bean in the spring bean configuration file of typeResourceBundleMessageSource or ReloadableResourceBundleMessageSource.
- For change of locale support, define localeResolver bean of type CookieLocaleResolver and configure LocaleChangeInterceptor interceptor. Example configuration can be like below:

```
<beans:bean id="messageSource"
    class="org.springframework.context.support.ReloadableResourceBundleMe
    <beans:property name="basename" value="classpath:messages" />
    <beans:property name="defaultEncoding" value="UTF-8" />
</beans:bean>

<beans:bean id="localeResolver"
```

```
        class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
        <beans:property name="defaultLocale" value="en" />
        <beans:property name="cookieName" value="myAppLocaleCookie">
    </beans:property>
        <beans:property name="cookieMaxAge" value="3600"></beans:property>
    </beans:bean>

    <interceptors>
        <beans:bean
            class="org.springframework.web.servlet.i18n.LocaleChange
    Interceptor">
            <beans:property name="paramName" value="locale" />
        </beans:bean>
    </interceptors>
```

- Use spring:message element in the view pages with key names, DispatcherServlet picks the corresponding value and renders the page in corresponding locale and return as response.

What are some of the important Spring annotations you have used?

Some of the Spring annotations that I have used in my project are:

@Controller – for controller classes in Spring MVC project.

@RequestMapping – for configuring URI mapping in controller handler methods. This is a very important annotation

@ResponseBody – for sending Object as response, usually for sending XML or JSON data as response.

@PathVariable – for mapping dynamic values from the URI to handler method arguments.

@Autowired – for autowiring dependencies in spring beans.

@Qualifier – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.

@Service – for service classes.

@Scope – for configuring scope of the spring bean.

@Configuration, @ComponentScan and @Bean – for java based configurations.

AspectJ annotations for configuring aspects and advices, @Aspect, @Before, @After, @Around,@Pointcut etc.

What are the advantages of JdbcTemplate in spring?

Less code: By using the JdbcTemplate class, you don't need to create connection,statement,starttransaction,commit transaction and close connection to execute different queries. You can execute the query directly.

Explain RowMapper Interface?

RowMapper interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection. So we don't need to write a lot of code to fetch the records as ResultSetExtractor.

Advantage of RowMapper :

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

Method of RowMapper interface

It defines only one method mapRow that accepts ResultSet instance and int as the parameter list. Syntax of the method is given below:

public T mapRow(ResultSet rs, int rowNumber)throws SQLException

Example:

```
public class EmployeeDao {

private JdbcTemplate template;


public void setTemplate(JdbcTemplate template) {

    this.template = template;

}


public List<Employee> getAllEmployeesRowMapper(){

 return template.query("select * from employee",new RowMapper<Employ
ee>(){

    @Override

    public Employee mapRow(ResultSet rs, int rownumber) throws SQLEx
ception {

        Employee e=new Employee();

        e.setId(rs.getInt(1));

        e.setName(rs.getString(2));

        e.setSalary(rs.getInt(3));

        return e;

    }   });

}

}
```

DownloadLinks JARS

Spring :
https://repo.spring.io/release/org/springframework/spring/

Hibernate :
https://sourceforge.net/projects/hibernate/files/hibernate3/

Struts :
https://struts.apache.org/download.cgi

RestFul WebService (Jersey):
http://repo1.maven.org/maven2/com/sun/jersey/jersey-archive/1.19/jersey-archive-1.19.zip

JAXB For XML (Marshalling and UnMarshalling):
https://jaxb.java.net/2.2.11/jaxb-ri-2.2.11.zip

Eclipse :
http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junosr2

Oracle DB (Oracle 10gXE):
http://www.mediafire.com/download/roc48z1i6iitpu4/OracleXE.exe

Oracle Type 4 :
http://central.maven.org/maven2/com/oracle/ojdbc14/10.2.0.2.0/ojdbc14-10.2.0.2.0.jar

MySQL:
http://central.maven.org/maven2/mysql/mysql-connector-java/5.0.5/mysql-connector-java-5.0.5.jar

Commons -logging:
http://central.maven.org/maven2/commons-logging/commons-logging/1.2/commons-logging-1.2.jar

Java Mail :
http://central.maven.org/maven2/javax/mail/mail/1.4.3/mail-1.4.3.jar

Commons-codec:
http://central.maven.org/maven2/commons-codec/commons-codec/1.10/commons-codec-1.10.jar