

RS = 150/-

SPRING NOTES

BY

SHEKHAR SIR

SATYATECHNOLOGIES

SRI RAGHAVENDRA XEROX

Software Languages Material Available

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199

Spring

2-2-2015

C, C++, Java ^{are} programming languages

JDBC, Servlet, JSP, EJB, are technologies.

Struts, Spring, hibernate are framework

- framework is a special software i.e built on technology having ~~to~~ the ability to generate common logic of the Appⁿ dynamically.
- framework is a special software that provides abstraction layer on existing technology ^{to} simplifies the application development process

Plane JDBC Appⁿ

Register ~~the~~ JDBC Driver }
Establish the conn. } (Common logics)

Create Statement object

Send & execute query } Appⁿ specific logics.
Gather & process result }

Execution handling } Common logics
close JDBC Objects }

Boiler Plate code problem

Spring-JDBC App

→ Get JDBC Templates class object.

→ Send & execute query.

→ Gather & process the results

While working with technology we should develop both Common & Appⁿ specific logic this use Boiler Plate code problem. writing same logic in multiple Appⁿ of some categories ^{rises} boiler plate code problem

→ While working with framework we just need to take care of Appⁿ specific ^{logic} development, because common logic of the appⁿ will be generated automatically by framework

there are 2 types of framework

- ① invasive framework :- Here the classes of framework API's extend or implement class or interface of framework API that means we can execute these classes outside the framework environment eg. Struts 1.x.
- ② non-invasive framework :- Here classes of framework API's do not implement or extend framework API interfaces, class that means we can execute these classes outside of the framework environment eg. Spring, hibernate, struts 2.x, JSF etc.

Diff. categories Java framework

- a) Web frameworks
→ provides abstraction layer on servlet, JSP, technologies & allows to developers MVC based web API's
eg. Struts, JSF, Webwork & etc --.
- b) ORM frameworks
→ provides abstraction layer on JDBC technology & allows to develop objects based O-R mapping persistence logic
eg. hibernate, iBatis, OJB & etc
- c) Application framework | Java JEE framework
⇒ Provides abstraction layer on multiple Java, JEE technologies (JDBC, servlet, JSP, EJB, JMS & etc) & allows to develop all kinds of applications.
eg. Spring.

Spring provides abstraction to all technology.

Spring

Type: Application framework

Version: 3.x | 4.x (compatible with Java 1.6+)

Vendor: Interface21

Creator: Mr. Rod Johnson

Open Source

To Download: download as zip file from www.springframework.org
www.io.spring.org.

Zip file: → Spring-framework-3.0.5.RELEASE-with-docs.zip

To install SW: Extract zip file

Online tutorials: roselindia, javatpoint, tutorialspoint.

Ref. Books: Pro Spring
 Spring in Action

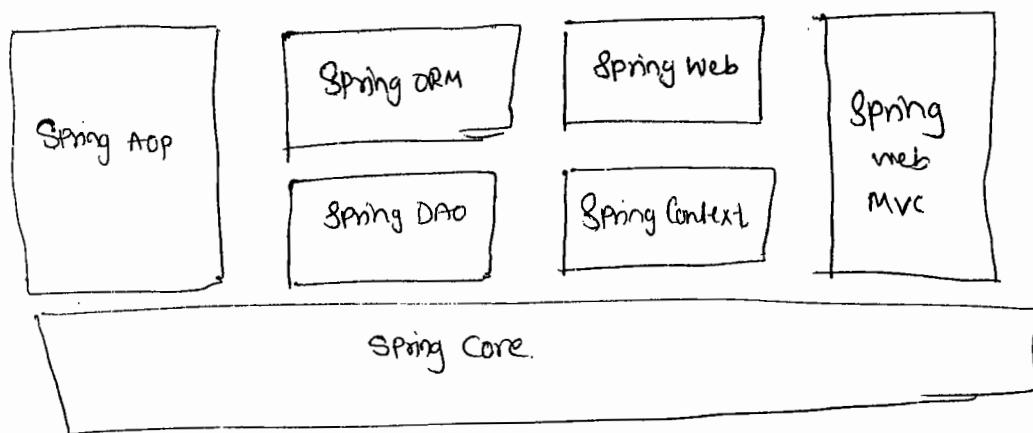
<SP_HOME>\dist → gives spring libraries (jar file)

<SP_HOME>\docs → gives api docs, reference doc

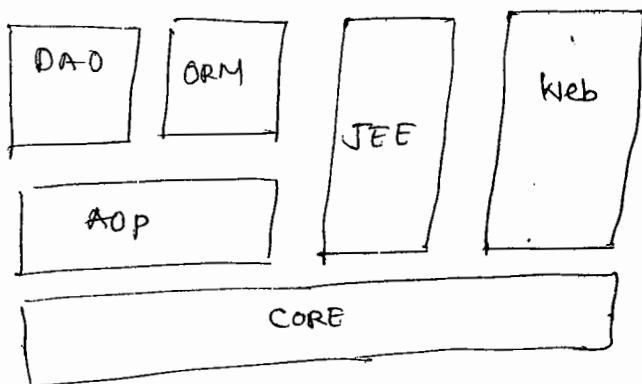
<SP_HOME>\Project → gives sample apps, projects.

<SP_HOME>\src → gives source code.

^{1. X} Spring framework overview.



Spring 2.X overview diagram



Spring 2.X Web = Spring 1.X Web + Spring 1.X Web MVC modules

Spring 2.X JEE = Spring 1.X Context module.

Spring Core: provides IOC containers to perform Dependency injections
(Assigning values to the resources dynamically)

Spring DAO → provides abstraction layer on JDBC to develop Persistence logic.

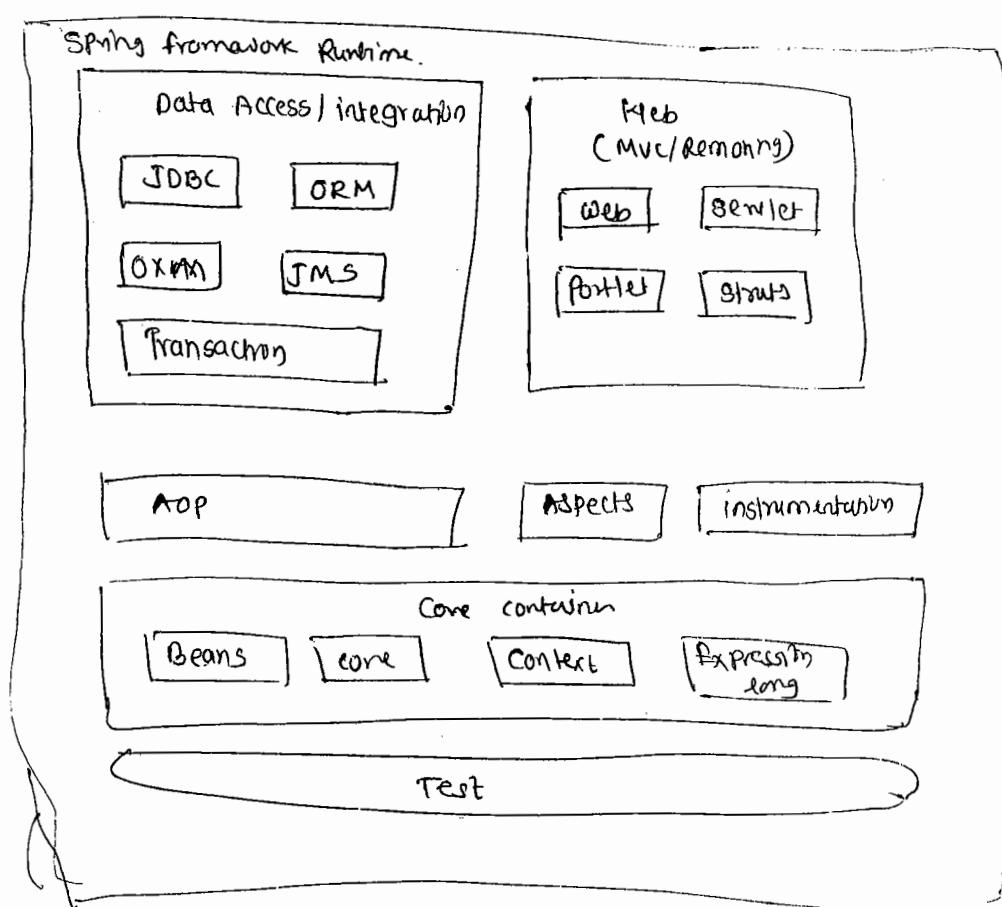
Spring ORM: Provides abstraction layer on ORM frameworks like hibernate, toplink and etc. to develop O-R mapping persistence logic

Spring Web: a) Provides Plugins for startst spring, Jsf + Spring integration
b) Provides spring webmvc as web fw to develop web Applications

Spring JEE: Provides abstraction layer on multiple Java, JEE technologies like Jndi, Jms, rm, ita, f etc. etc. to develop JEE apps.

Spring AOP: Provides a different mechanism or methodology to develop & apply middleware services like logging, security, Transaction etc.

B-X Overview diagram



Defⁿ Spring is an open source, light weight, non-invasive, loosely coupled aspect oriented, dependency injection based, Java API framework to develop all kinds of applications.

- along with spring we get its source code this makes spring as ~~lightest~~ open source.
- Spring is lightweight Reasons,
 - ① Spring framework size is very less
 - ② Spring container can be activated as in memory containers without any physical server support
- Spring resources can be developed without spring API
- Spring is non-invasive framework, because the classes of spring app can be developed without extending or implementing spring api classes / interfaces

4-2-2015

- Spring framework is not replacement or alternate for JSE, ~~JEE~~ JEE technologies, fact it is compliment to these of technologies.
- Spring framework is a loosely coupled because
 - a) we can use one or another module of spring without having dependency with other modules.
e.g. we can use spring core + jdbc modules to develop persistence logic without using other modules.
 - b) the dependent value of spring resource can be configured through XML file without hardcoding them directly in resources (classes)
 - c) we can use only spring to develop complete project or we can use spring along with other frameworks or java technologies to develop ~~with~~ Project.

- Aspect oriented programming is a methodology of developing middleware services logic separately from main stream business logic and linking them with business logic dynamically at run time.
- middleware services are like security transaction etc.. since spring gives support for this AOP, we say spring is Aspect oriented.
- if the underlying server or container or framework or runtime environment dynamically assign value to the resources then it is called dependency injection. in this underlying server or container pushes the value/resource. ex- the way Jvm called constructor to initialise object when object is created is called as dependency injection.
eg: if course material assigned to student the moment he registers for course is called dependency injection.

there are two modes dependency injections.

- a) Setter injection
- b) Constructor Injection

- while developing project generally come across diff. types of classes
 - a) Pojo class: the class that can be executed in jdk environment without taking support of third party libraries is called Pojo class
 - b) Java Bean: the class that follows some standards is called Java bean. this class does not contain business logic but contains accessor methods. (Setter & Getter) the standards are
 - a) must be a private class
 - b) must have a private member variables (Properties)
 - c) must have public getter & setter methods for each property
 - d) must have 0-param constructor directly or indirectly
 - e) must implement java.io.Serializable

Note: → Java Bean is a Pojo class, but every Pojo class need not be a Java Bean.

- c) Component class/Bean class:- the class that uses its member variable in its member methods while developing business logics is called component class/ Bean class.

- If bean class is managed by Spring Container then it is called Spring Bean.
- If bean class is managed by EJB Container then it is called EJB Bean.
- bean class can be developed as POJO class or as non-POJO class.

class Test {

```
int a;
int b;
Public void bind() {
    -- b-logic using Jdk api
}
```

}

}

"Test" is POJO class & Bean class / component class

Public class Test implements Serializable {

```
private int sno;
private String name;
// write setter & getter methods.
```

g

"Test" is Java bean & Pojo class.

Public class Test {

int age;

Public int calcPension() {

-- uses Spring API for b-logic

3

"Test" is not a POJO class. Test is Bean class.

Container is software appn ^{that} can manage whole lifecycle of given resources it is like an aquarium. Spring framework gives us a built container as light weight

- in memory container without having support of any physical server like Tomcat

There are

a) Beanfactory Container (basic container)

b) Application Context Container (Extension of Beanfactory Container)

⇒ both these containers manage & executes Spring beans & also capable
performing dependency injection.

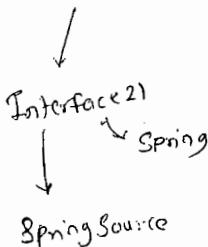
Spring

9-2-2015

Q. Why framework?

- ① When Java is released, there are no predefined API's for connecting with database or for connecting with mail server, or transaction management etc.
- ② In early days of Java, a programmer itself started coding everything in a project, it made burden on the programmer.
- ③ to reduce the burden sun microsystem release a set of API's for connecting with databases, mail server, transaction etc. with a name J2EE
- ④ J2EE has lots of API and it is more burden on developer because remembering & integrating J2EE API's in a project is complex.
- ⑤ to get out of burden on directly working with J2EE API's, 3rd party vendors started providing framework in Java.
- ⑥ framework are considered as a top layers (Abstraction layers) on top of J2EE API's
- ⑦ the following are the benefits of directly working with frameworks:
 - ① frameworks remove the burden of writing repeated code, so burden of the programmer is reduced.
 - ② frameworks will help to finish a project as early as possible so projects can be delivered fastly to the clients.
 - ③ A project can be maintained easily it means a project with minimum changes, we can add new feature to the project.

" Rod Johnson



J2SE → Java API

10-2-2015

J2EE → API's

↓
Servlet API, JSP API, JDBC, JNDI, EJB, JTA, JMS

Spring → Modular framework

the below are the major reasons for choosing Spring framework by the industry.

- ① Spring framework can be used to develop all layers of an application it means we can develop core Java application, Web application, Remoting applications etc. using Spring framework
- ② Spring framework is versatile framework it means we can easily integrate our Spring application with application developed using other framework.
- ③ Spring framework suggesting to use ordinary Java classes and interfaces for creating an application. This is also called PoSI PoJO model.
- ④ Spring framework has given its own container called IOC container (Inversion of control) so we no need to depend on third party containers provided by the third party vendors.

Q What is a Non-invasive framework?

A → ① If framework is not forcing programmers to extend the a class from a Superclass or a Interface which is given by a framework then it is called a non-invasive fw.

② If a framework is forcing programmers to extend or implement a Java class from a Superclass or interface given by framework, then it is called invasive framework.

for eg: Struts { Spring & hibernate ^{& JSF & Struts 2.x} → non-invasive framework.
Struts 2.x → invasive framework.
struts 2.x → non-invasive framework}

Q Why Spring fw is called light weight framework?

Ans → Spring framework is divided into six modules a programmer can choose particular module only to develop a spring application, it is not compulsory to choose all the module to develop of an app? so Spring framework is called a light weight framework

11-2-2015

Spring → Modular framework.

- light weight
- non-invasive.

Modules

- | | |
|---------------|---------------------------------------|
| ① Spring core | ④ Spring JEE (services) |
| ② Spring AOP | ⑤ Spring DATA Data Access. |
| ③ Spring MVC | ⑥ Spring test |

Modules

- ① Spring core
- ② Spring AOP
- ③ Spring MVC
- ④ Spring JEE (services)
- ⑤ Spring Data Access
- ⑥ Spring Test

① Core module:

- this core module is the fundamental to the Spring framework
- In Spring, all the remaining modules are developed on the base Spring core module.
- in core module we can get the following information

- ① How to configure the application classes in spring configuration file.
- ② How to utilize the dependency injection mechanism
- ③ How to obtain a Spring IOC container
- ④ How to modify object graph at runtime using Spring Expression language. (Aspect) (SPEL)

Q. AOP (Aspect Oriented Programming):

P C withdraw

{

P v withdrawMoney (acno, amount)

{

// Security → cross-cutting concern, Service logic, Secondary logic.

// B-logic → Primary logic

{

}

- When developing application, the business logic which are created or called Primary logic.
- In order to make Business logic as efficient, some Service logic are attached to ~~new~~ Business logic. These Service logic are called Secondary logics.
- In Spring framework Secondary logic are also called cross-cutting concern.
- In a Project, if Primary & Secondary logic are ~~combined~~ combinedly developed, then we loose reusability of the services to the multiple primary logic.
- In Spring framework, Aop module is given ~~used~~ for separating the primary and secondary logic and ~~then~~ injecting or combining secondary logic to the primary at run time.

3. MVC

- this MVC module is given to develop
 - ↓ Presentation layer (Web layer) in a project by following MVC architecture.
- this MVC module provide all the infrastructure like a controller, tag libraries, annotation etc. for creating web layer of a project.
- this MVC module of the Spring framework is almost equal to Struts framework.

4. Spring JEE (Services)

AOP tells to how to separate Business logic & Service logic.

JEE tells to implementing service logic.

implementing middleware services.

↓
transaction, security, mailing, messaging

12-2-2015

Core modules

configure.

Servlet → web.xml.

Spring beans → xml
(POJO's)

- this module of the Spring framework tells a programmer about How to implement the middleware services to the Business logic of Java project.
- middleware services are transaction, security, mailing, messaging, scheduling etc.
- the difference between Spring AOP module and Spring JEE module is,
AOP module tells How to separate B-logic & services and How to injecting or combining at run time. but JEE module tells about How to implement Services.

Spring Data Access module.

① Spring-JDBC

↓
abstraction layer
on top
of JDBC tech.

② Spring ORM

→ this module is divided into two submodule.

① Spring-JDBC

② Spring-ORM

→ Spring JDBC module is an abstraction layer on top of JDBC technology

this layer avoids the Boiler Plate code used in JDBC Programming

→ Spring ORM module is an abstraction layer on top of ORM tools.

→ When working with ORM tools like a hibernate again we have a

Boiler plate code this Spring ORM layers avoids boiler plate code of
ORM tools.

Spring Test Module.

* difference bet Mock object & proxy object ?

Junit Tool
TestNG
JMeter } Testing tools for testing Mock object is necessary
which is created by Spring f/w

- this module is given for conducting unit testing on the Spring code
- Unit testing is the primary responsibility of a Java programmer.
- To conduct unit testing we use a tools like JUnit, TestNG, or JMeter etc.
- While testing by creating test cases sometimes we need Mock objects.
- The benefit of this Spring test module is framework it self creates a mock objects.
- In real time, this Spring test module will be used in very less no. number of cases.

SPRING - CORE Module

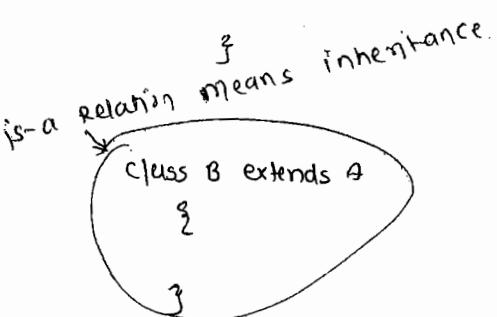
18-2-15

Tight coupling & loose coupling.

Why has-a relation preferred than is-a
or better of

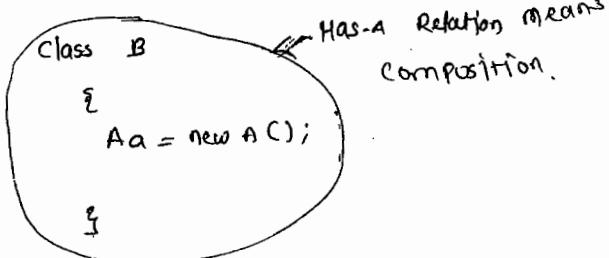
Ex. ~~Car~~ is a vehicle

Class A {
 M1();



Ex. car has-a engine.

Class A {
 M1();
 M2();



(class Emp). extends Addr {

→ Tight coupling & loose coupling between objects.

Q) Why HAS-A relationship is better than IS-A relationship?

A → ① Reason-1

→ If we apply IS-A relationship between two classes then one object of subclass can get functionality of a super class for only once. It is not possible to get functionality for more than once.

→ If we apply Has-A relation then it is possible to get functionality of one class in an object of another class.

→ For e.g. we have class Employee extends Address class. Here one employee object can get functionality of

address class ~~can~~ for once only.

→ if we create Address class object in Employee class then it is possible to get functionality of the Address into Employee for multiple time

class Employee extends Address (IS-A)

{

}

→ Class Employee ~~HAS-A~~ (HAS-A)

{

Address temporary = new Address();

Address Permanent = new Address();

}

Reason: 2

→ if we apply Is-a relationship then all functionality of Superclass will be ~~can~~ inherited into subclass. If we want some functionality the remaining functionality should be made as a private. If another subclass want private functionality of super class then it is not possible

→ If it is a HAS-A relationship then we no need make the functionality of a class as a private. We can call only required functionality by using an object of the class

class B

{

A a = new A();

void x()

{

a.m1();

a.m4();

}

}

class C

{

A a = new A();

void y()

{

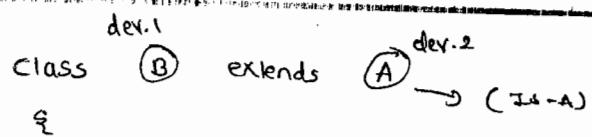
a.m2();

a.m3();

}

}

Reason 3



3

dev.1 (B) \rightarrow testing

A \rightarrow tested automatically.

Class B

2

A a = new A();

3

(mock object) \rightarrow possible in HAS-A
 \rightarrow not in IS-A.

- In IS-A Relation, when we are testing the subclass, its superclass is also be tested.
- If any problem occurs in a superclass then subclass testing ~~will fail~~ will be failed.
- In case of HAS-A relationship, we can create & inject mock object of the class into another class and then we can test that class only.
So for unit testing also HAS-A relation is better than IS-A relation

Class Travel $\xrightarrow{\text{caller class}}$

{

$\text{Car c = new Car();}$

void Journey()

{
 c.go();
 c.move();
 }

Class Car $\xrightarrow{\text{dependent class}}$

{

void go() move()

{

{

Connection (Car)

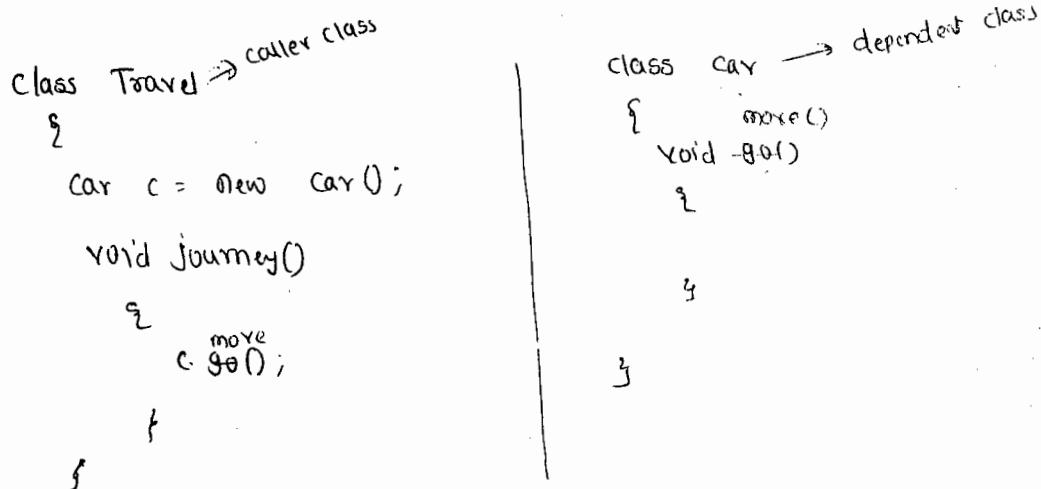
Reference Variable which Pointing the object of implemented class

\rightarrow returns the object of implemented class

tight coupling

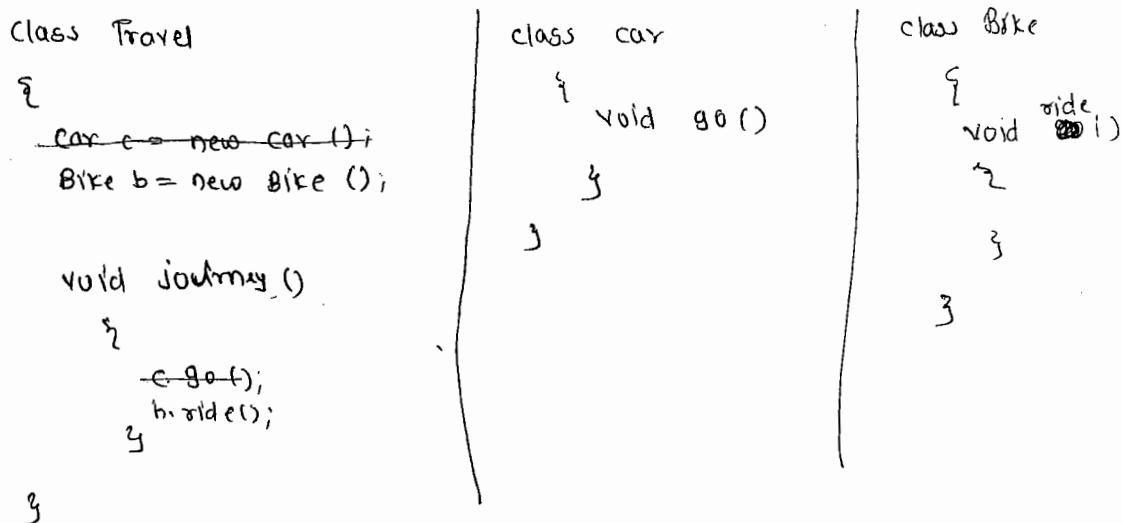
→ tight coupling b/w two classes will occur in the following situation

- ① If functionality name is changed in the dependent class name
(method)
- then we need to modify caller class also.



→ In the above Travel class is depending on car, in car functionality name changed to move so in Travel class also functionality name is changed to move. It is a ^{need} tight coupling.

- ② if a caller class want to change its dependency to another similar type of class then also we need to change or modify caller class.



→ In the above when we want to change dependency from car class to Bike class then we have changed caller class ~~Travel~~ called Travel. this is also called tight coupling.

Loose Coupling

17-2-2015

→ loose coupling b/w two classes can be achieved in the following two ways.

- ① by following POJO/POJ~~O~~ model
- ② by applying dependency injection mechanism.

IoC → Inversion of control
↓
3rd person

- By following POJO/POJ~~O~~ model, a group of similar classes are going to have common functionality names.
- By taking the reference variable of an interface, we can store any one of its implementation class object.
- Inversion of control indicates the an external classes managing the classes of an application
- an external class means like a container manages the collaboration between or dependency b/w classes of application.
- IoC is of 2 types
 - ① Dependency look-up
 - ② Dependency injection.
- In Dependency look-up, a class explicitly ask the container to get its dependent object or to take its dependent object.
- In Dependency injection, a container only injects the dependency to a class
- Spring fw promotes dependent injection^{type} of IoC.
- ① by following POJO/POJ~~O~~ model.
② by using dependency injection mechanism. we can get the loose coupling b/w the 2 classes or object.

Types of Dependency injection.

- In Spring fw there are 4 types of Dependency injections
 - ① constructor injection.
 - ② setter injection.
 - ③ interface injection.
 - ④ lookup method / method injection.
- Constructor and setter injection are used very frequently for Injecting the dependencies.
- Interface & lookup method injection are used very rarely in spring Application.
- If a dependency object is injected to the caller by calling a constructor in the caller class, then it is called constructor injection.
- If a dependency object is created to a caller by calling setter method defined in a caller class, then it is called setter injection.

public class Travel

{
 private Vehicle v;

 public Travel (Vehicle v) {
 called constructor
 }

 this.v = v;

}

 public void journey () {
 =

}

g

}
constructor
injection

- ⇒ In the above class a constructor is defined to inject its dependency so it is called constructor injection.

```

public class Travel
{
    private Vehicle v;
    public void setV(Vehicle v)
    {
        this.v = v;
    }
    public void Journey()
    {
        ...
    }
}

```

→ in the above class a setter method is defined to inject a its dependency so it is called **Setter injection**.

POJO, Java Beans, Spring Beans

- a Pojo class means it is a class which doesn't exceed the boundary of Java API.
- no exceeding the ^{boundary of} Java API means a class be compiled directly with Java compiler by without taking the support of any jar file.
- a Java Bean is also a Java class which follows the below rules.
 - ① class must be a public class.
 - ② class must contain default constructor.
 - ③ a private variable should contain either setter, getter or both methods.
 - ④ a class can almost implements java.io.Serializable interface.

Ex. 1

```

class A → POJO ✓
{
    int k;
    A (int k)
    {
        this.k = k;
    }
}

```

Ex-2

```
public class A
{
    private int k;
    public A (int k)
    {
        -----
        }
    public void setK (int k)
    {
        -----
        }
}
```

Java Bean ✗ no default constructor.
POJO ✓

Ex-3

```
public class A
{
    private int k;
    public A ()
    {
        -----
        }
    public void setK (int k)
    {
        -----
        }
}
```

Java Bean ✓
POJO ✓

Ex-4

```
public class A extends Exception
{
    -----
}
```

POJO ✓

Java Bean ✗ because extending Exception class.

Ex-5

```
public class A
{
    -----
}
```

POJO ✓

Java Bean ✓

Ex. 6

Public class A implements Serializable

```

{
    void m1()
}
=
```

Pojo ✓
JavaBean ✓

}

Ex. 7

Public class MyServlet extends HttpServlet

```

{
    =
}
```

Pojo X
Java Bean X

}

* Every Java Bean class is definitely Pojo class also but every Pojo class is not a Java Bean class.

→ As part of a Spring Application, whatever the classes we developed they are called Spring Beans.

→ In most of the cases a Spring Bean is also a Pojo. Sometimes Spring Bean is Java Bean also

→ In exceptional cases a Spring bean will be not a Pojo or Java Bean

Ex. 1

Public class A

```

{
    private int i;
```

Pojo ✓
JavaBean ✓

Public void setI (int i)

Spring Bean ✓

```

{
    this.i = i;
}
```

}

3

Ex.2

```
public class A  
{  
    private int i;  
    public A (int i)  
    {  
        this.i = i;  
    }  
}
```

Pojo ✓

Java Bean ✗

Spring Bean ✓

Ex.3

```
public class A implements InitializingBean  
{  
    ...  
}
```

Initializing Bean

↳ spring framework interface.

Pojo ✗

Java Bean ✗

Spring Bean ✓

When Beanfactory Container is activated the entries of Spring Configuration file will be verified by using SAX XML Parser
↳ Sample API for XML

Object obj = factory.getBean("db");

Bean id

- ① Beanfactory Container to load "Bean" class based on the given Bean id "db" using configurations done in spring cfg file.
- ② makes Beanfactory Container to create "DemoBean" class object using -Param Constructor.

factory.getBean("db"); method returns "~~DemoBean~~" Bean class object and we are referring that object using java.lang.Object class with ref. "obj"

Note :- getBean() method is the Predefined method in predefined class "Object" If we want to get the Object from Beanfactory Container we use this getBean() method. If we call Business method, that method is not part of the predefined class Object. It is part of the implement class of ~~Spring Interface~~ SpringInterface, so we must typecast i.e

(classname) obj = (classname) Obj;

Naming Convention.

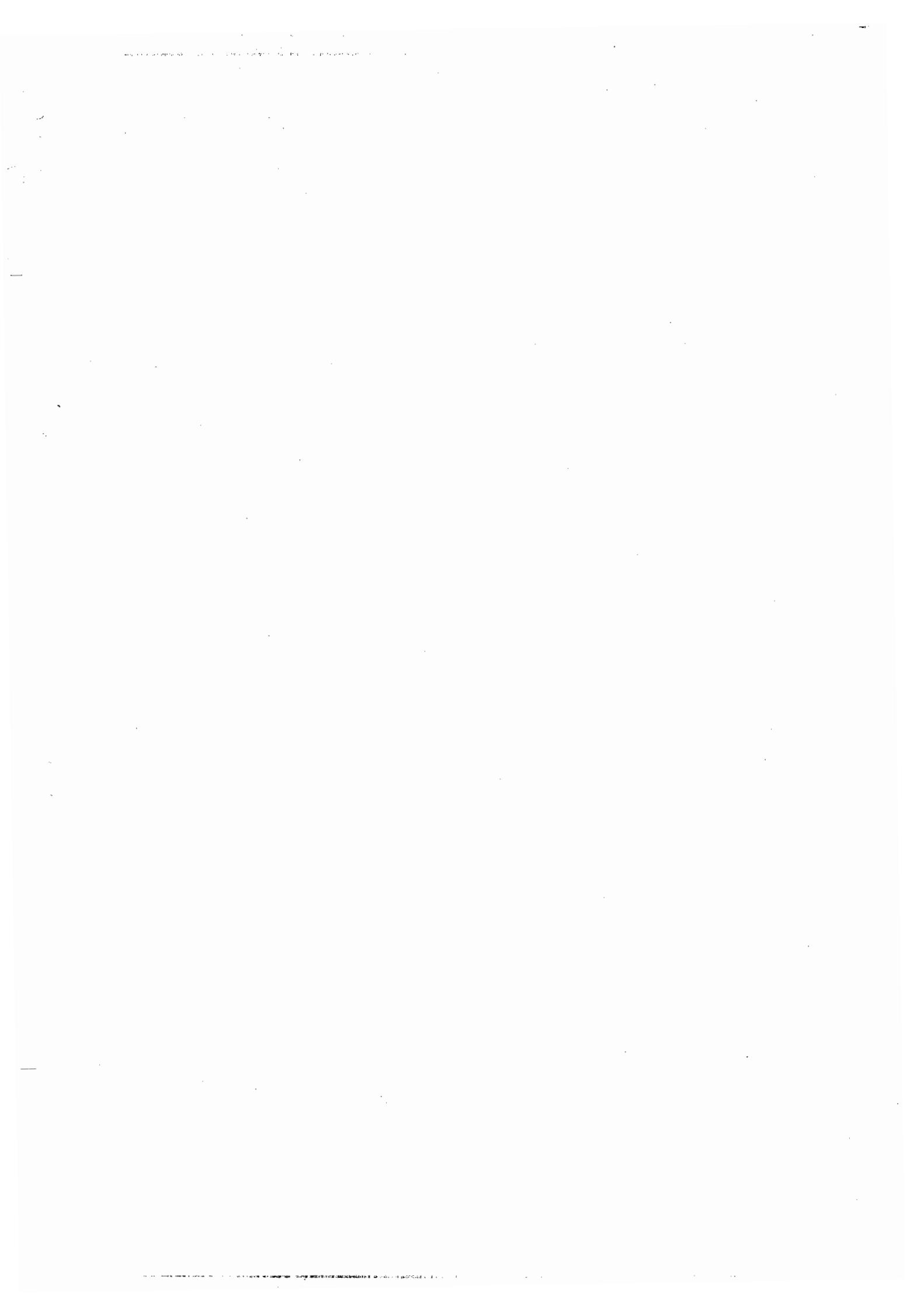
X → Spring interface name.

XBean → Spring Bean class name.

X.cfg.xml → Spring cfg file name.

Xclient → client APP name.

for Example Put X → Test



Spring Configuration file

18-2-2015

Anyname¹⁵.xml

demo.xml

```
<beans>
  <bean id = "id1" class = "com.sathyam.beans.Travel">
  =
</bean>
<bean id = "id2" class = "com.sathyam.beans.Car">
  =
</bean>
</beans>
```

→ only class is configured using Spring XML tag not interface.

/ class may be concrete, abstract, final

~~Set → can't be duplicated for same~~

- ① In order to tell classes of our application to the Spring IOC Container we configure our classes in spring configuration file.
- ② a Spring Configuration file is an XML file and the it can be named as Anyname.xml
- ③ if we write information about our classes to the another file or then it is called configuration.
- ④ In web applications we configure our servlet classes in deployment descriptor file (web.xml). similarly in Spring application we configure Spring beans in a Spring Configuration file.

- ⑤ a spring configuration file contains root element as <beans> & each class is configured using <bean> tag

<anyname.xml>

<beans>

<bean id = "xxx" class = "fully qualified classname">

|||

</beans>

<bean id = "xxx" class = "fully qualified classname">

|||

</bean>

</beans>

Important Statement

- ① ~~as~~ using <bean> tag, only classes ^{can be} ~~are~~ configured but not interfaces
- ② a class can be concrete class, abstract class, or it may be final class.

<bean id = "id1" class = "com.satyendra.beans.Travel" > → valid
↓
concrete class

<bean id = "id2" class = "com.satyendra.java.util.Calendar" > → valid
↓
abstract class

<bean id = "id3" class = "java.lang.Runnable" > invalid
↓
interface

- ③ we can configure same class for multiple times with different ids

<bean id = "id1" class = "com.satyendra.beans.Travel" > ✓

<bean id = "id2" class = "com.satyendra.beans.Travel" > ✓

- ④ id attribute follows

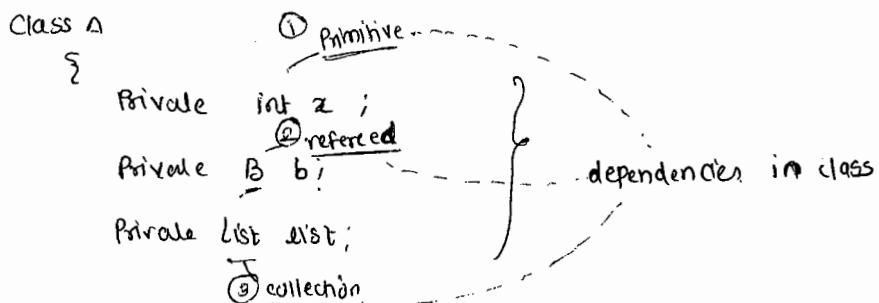
- ④ Id attribute follows XMLE id rules.
- i) same id should not be duplicated in entire XML.
 - ii) id should not contain special characters. It can have only alphanumeric characters.

<bean id = "id1" class = "com.sathyam.beans.Travel"/>

<bean id = "id1" class = "com.sathyam.beans.Travel"/> → Invalid
 ↳ duplicated hence Invalid.

<bean id = "@id1" class = "com.sathyam.beans.Travel"/> → Invalid. because
 ↳ Special character.
of special character.

<bean id = "lid" class = "com.sathyam.beans.Travel"/> → valid



Types of dependencies in a bean

→ ① a bean can have ^{a maximum of} 8 types of dependencies

- ① primitive type.
- ② referenced type.
- ③ collection type.



class A

```
{  
    private int x; → Primitive type  
    private B b; → reference type  
    private List list; → Collection type.
```

}

- ② In a bean class if we defined a constructor for injecting a dependency then we need to configure <constructor-arg> tag in a xml file.
- ③ If a setter method is defined for injecting a dependency then we need to configure <Property> tag in a XML file

Example 1

P C A

```
{  
    private int x;  
    public void setX (int x) → setter method.
```

```
{  
    this.x = x;  
}
```

}

when SpringContainer is activated by giving this spring cfg file the Spring container creates A class object having "id" as the object Name & calls the method.

demo.xml

```
<beans>           id / ref. name  
    <bean id = "id" class = "A">  
        * <Property name = "x" value = "100" />  
    </bean>           Bean class  
</beans>           Bean Property  
                                Value to be injected
```

Ex.2

P C A

{

Private int x;

Public A (int x)

→ constructor.

{

this.x = x;

t

3

demo.xml

<beans>

<bean id = "id1" class = "A">

<constructor-arg value = "100" />

</bean>

</beans>

While configuring Spring Bean if you place only <constructor-arg> tag under <bean> tag then Spring container uses Parameterized Constructor to create Spring Bean class object & to perform constructor injection on bean properties.

Note: In <Property> tag name attribute is mandatory. but in <constructor-arg> tag name attribute is not allowed.

Ex.3

demo.xml

P C A

{

Private int x;

Private int y;

Public A (int x)

{

this.x = x;

3

Public void setY(int y)

{

this.y = y;

3

4

<beans>

<bean id = "id1" class = "A">

-><constructor-arg value = "100" />

-><property name = "y" value = "200" />

</bean>

</beans>

Bean with Multiple Constructors

19-2-2015

- a spring bean can have more than one constructor, because a constructor can be overloaded.
- if bean class contain many constructor then we need to configure bean class for multiple times in xml file.

for example

in this Example dependency is Primitive type.

```
int x;  
int y;
```

Public class A

{

 private int x;

 private int y;

 Public A (int x) // with 1 argument

{

 this.x = x;

}

 Public A (int x, int y) // with 2 argument

{

 this.x = x;

 this.y = y;

}

g

Spring.xml

<beans>

 <bean id = "a1" class = "A">

 <constructor-arg value = "100"/>

 </bean>

 <bean id = "a2" class = "f">

 <constructor-arg value = "50"/>

 <constructor-arg value = "70"/>

 </bean>

</beans>

dependency is
Value → indicates a primitive type
ref → indicates dependency is reference type.

dependency of reference type.

- if a dependency of bean class is reference type the IOC container injects object.
- In the configuration file we use "ref" attribute to configure the referenced type of dependency.

Ex:

```
public class A
{
    private B b;
    public void setB(B b)
    {
        this.b = b;
    }
}
```

Xml file

```
<beans>
    <bean id = "id1" class = "A">
        <property name = "b"
            ref = "id2"/>
        ↳ indicates injects object as dependency is reference type
    </bean>
    <bean id = "id2" class = "B"/>
</beans>
```

Note ① <Property> tag has the below attributes

- ① name
- ② value
- ③ ref

- ② name attribute is mandatory, in value & ref, we can use only one at a time.

* <ref> tag:

- when dependency is reference type then in xml file we can configure ref as attribute or ref as tag.
- if we configure ref as tag then it should have either local or parent or bean has an attribute.

- ① → if we configure local attribute with <ref> tag then the Spring container will search for dependent bean in same xml.

Example

Spring.xml

```
<beans>
  <bean id = "id1" class = "A">
    <property name = "b">
      <ref local = "id2"/>
    </property>
  </bean>
  <bean id = "id2" class = "B"/>
</beans>
```

- ② If we configure the Parent attribute with <ref> tag then Spring Container checks for dependent bean in Parent xml

Example

Parent.xml

```
<beans>
  <bean id = "id2" class = "B"/>
</beans>
```

child.xml

```

<beans>
  <bean id = "id1" class = "a">
    <property name = "b">
      <ref parent = "id2"/>
    </property>
  </bean>
</beans>

```

- ④ → If we configure bean attribute with `<ref>` tag then Spring Container will search for dependent bean in the local XML, if not found then container will search for dependent bean in Parent XML.
- If we use `ref` as attribute then it will be equal to `<ref>` tag with `bean` attribute.

class B provide runtime support to class A
Hence it is container class

```

Class A
{
  void m1()
}

```

```

Class B → container class
{
  public void m1(String[] args)
  {
    A a = new A();
    a.m1();
  }
}

```

Ioc container

- ① Beanfactory (I)
- ② ApplicationContext (i)

XML Beanfactory (C)

Beanfactory factory = new XML Beanfactory();

Spring Container

- ① Beanfactory interface (super)
 - ↑ extends
 - ② ApplicationContext interface (sub) → is better than Beanfactory interface
- ① Beanfactory factory = new XmlBeanfactory(); by creating its object SPRING IOC container started in an application
- ② ApplicationContext ctx = new ClassPathXmlApplicationContext();
- ③ Object getBean (String id); → returning object of particular class in Object format so we need type cast
- ④ Object o = factory.getBeans ("id2");
- ⑤ A. a = (A) o;

Spring Container

- a container is a class which provides runtime support to other classes
- if we create class with a main method then it will provide the runtime support to the other classes of a console Application. So a main class is called as container
- In AWT Programming a frame class provides runtime support to other classes Button, Label, etc. so frame class is a container class
- in Spring framework a Spring IOC Container means, it is an object of implementation class of either BeanFactory interface or ApplicationContext interface.
- ApplicationContext interface is a sub interface of a Beanfactory, so comparatively ApplicationContext container is better than Beanfactory.
- XmlBeanfactory is an implementation class of Beanfactory interface so if we create an object of XmlBeanfactory then Spring IOC container started in an application.

```
Beanfactory factory = new XmlBeanFactory();
```

- ClassPathXmlApplicationContext is an implementation class of ApplicationContext interface. so if we create an object of this implementation class then Spring Ioc container is started in application.

```
ApplicationContext ctx = new ClassPathXmlApplicationContext();
```

- When creating Spring Ioc container object we need to pass Spring configuration file as a parameter, so that Spring Container reads Bean definition from XML file.

- While creating Beanfactory object container object, we need to create Resource object for XML file and then we need to pass a resource object as parameter.

```
interface  
Resource res = new ClassPathResource("demo.xml");  
Implementation class
```

```
Beanfactory factory = new XmlBeanfactory(*res);
```

- When creating ApplicationContext container object we can directly pass XML file name as a parameter.

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("demo.xml");
```

- To call methods of Bean class from a main class we need to follow below steps.

- ① We need to Create a Spring Ioc Container Object
- ② We need to get the object of bean based on its id from Spring container object.

③ we need to invoke/call methods of the bean.

Spring Software

- a framework software is a group of jar files not a setup file
- To download Spring software we need to visit the following website

repo.spring.io/releases/org/springframework/spring/

21-2-2015

FirstApp

→ DemoBean.java
Spring-Config.xml
main.java

XSD file
nameSpace → element, attributes,
XMLS → keyword to import namespace into the
xml file.
BeanNamespace.

path → related to OS i.e. exe file.

classPath → related to Java.

Java reads classpath

→ to run .exe OS is needed, so place in
path

// DemoBean.java

```
Public class DemoBean  
{  
    Private String msg;  
    Public void setMsg (String msg)  
    {  
        this.msg = msg;  
    }
```

```
    Public void display ()  
    {  
        System.out.println (msg);  
    }
```

g;

Spring - config.xml

```
<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-4.0.dtd">

    <bean id = "demoBean" class = "DemoBean">
        <property name = "msg" value = "Welcome to Spring"/>
    </bean>
</beans>
```

// Main.java

```
import org.springframework.core.io.Resource;
import org.springframework.core.io.ClassPathResource;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;

class Main
{
    public static void main (String [] args)
    {
        Resource res = new ClassPathResource ("spring-config.xml");
        BeanFactory factory = new XmlBeanFactory (res);
        Object o = factory.getBean ("demoBean");
        DemoBean db = (DemoBean) o;
        db.display ();
    }
}
```

- DemoBean.java is a POJO class so it can be compiled directly.
- In main.java we have imported and used Spring framework API, so when we compile we will get compilation time errors. the reason is java compiler doesn't know Spring-framework API
- In order to resolve the errors we need to set following 3 jar files in classpath
 - ① Spring-core-4.1.2.RELEASE.jar
 - ② Spring-beans-4.1.2.RELEASE.jar
 - ③ commons-logging-1.0.4.jar

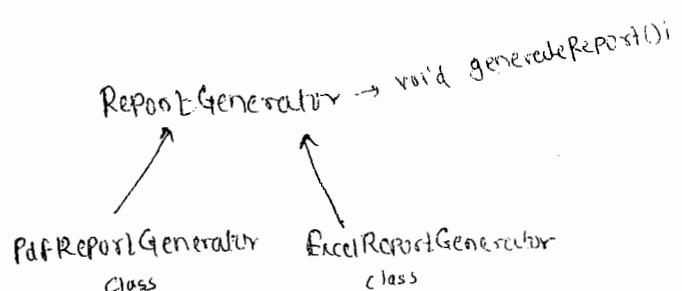
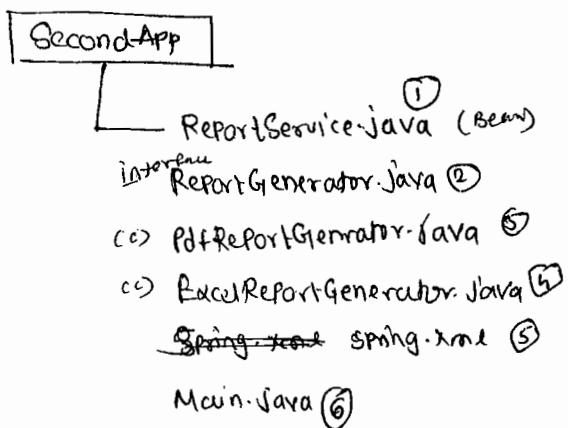
> javac *.java

> java Main

app → Welcome to Spring

Second Example

- in this example, we have a class ReportService and it depends on another object of type ReportGenerator.
- for ReportGenerator interface, we are creating 2 implementation classes PdfReportGenerator & ExcelReportGenerator.



1) ReportService.java

```
public class ReportService
{
    private ReportGenerator rg; interface
    public void setRg ( ReportGenerator rg )
    {
        this.rg = rg;
    }

    public void showReport()
    {
        rg.generateReport();
    }
}
```

1) ReportGenerator.java ✓

```
public interface ReportGenerator
{
    void generateReport();
}
```

1) PdfReportGenerator.java ✓

```
public class PdfReportGenerator implements ReportGenerator
{
    public void generateReport()
    {
        System.out.println ("Report in Pdf format");
    }
}
```

// ExcelReportGenerator.java

```
Public class ExcelReportGenerator implements ReportGenerator  
{  
    Public void generateReport()  
    {  
        System.out.println("Report in Excel format");  
    }  
}
```

[Spring.xml]

```
<beans
```

```
=>
```

```
<bean id = "reportService" class = "ReportService">
```

```
    <Property name = "rg" ref = "Pdf"/>
```

```
</bean>
```

```
<bean id = "Pdf" class = "PdfReportGenerator"/>
```

```
<bean id = "excel" class = "ExcelReportGenerator"/>
```

```
</beans>
```

[// Main.java]

```
Class main
```

```
{  
    main
```

```
    Resource res = new ClassPathResource("Spring.xml");
```

```
    BeanFactory factory = new XmlBeanFactory(res);
```

```
    Object o = factory.getBean("reportService");
```

```
    ReportService rs = (ReportService) o;
```

```
    rs.showReport();
```

```
}
```

```
}
```

Output → Report in Pdf format

```
Str= theList.get(class().getName())
System.out.println(str);
```

Dependency In the form Of a Collection.

- in a bean class if we take dependency as collection type of reference variable then internally spring IOC container creates a collection objects and injects the dependency.
- In a Spring Bean class we can take the dependency as any one of the following 4 types of collection.
 - 1) java.util.List
 - 2) java.util.Set
 - 3) java.util.Map
 - 4) java.util.Properties.
- In Spring configuration file to configure collection type we use the following tags
 - List → <list>
 - Set → <set>
 - Map → <map>
 - Properties → <props>

① <list> tag:

- ① this tag can be configure in xml, for reference variable of the following tags
 - 1) java.util.List interface
 - 2) java.util.Vector class
 - 3) java.util.ArrayList class
- ② If dependency of type List or ArrayList then internally IOC container creates and injects ArrayList class object
- ③ If dependency is Vector then container creates Vector class object.
- ④ Under <list> tag, we can configure sub-elements <value> and <ref>

Example

```
public class A  
{  
    private List theList;  
  
    public void setTheList(List theList)  
    {  
        this.theList = theList;  
    }  
}
```

Spring.xml

<beans>

```
<bean id = "a" class = "A">  
    <property name = "theList">  
        <list>  
            <value>100</value>  
            <value>Satya</value>  
            <value>100</value>  
            <ref bean = "b"/>  
        </list>  
    </property>
```

```
</beans>  
<bean id="b" class="B"/>  
</beans>
```

→ internally spring container will do the following work

```
A a = new A();  
List theList = new ArrayList();  
theList.add(100);  
theList.add("gautham");  
theList.add(100);  
  
B b = new B();  
theList.add(b);  
a.setTheList(theList);
```

② <set> tag

→ we configure this <set> tag in xml file for the following tags.

- 1) java.util.Set
- 2) java.util.HashSet
- 3) java.util.LinkedHashSet

→ for all the above 3 types, internally spring container creates ~~linked~~ LinkedHashSet object.

→ under <set> tag we can configure sub-elements as <value> & <ref>

→ if ~~no~~ a duplicate value configured then it will not be added to Set. It means container ignores the duplicates but doesn't throw an exception.

Ex:

```
public class A  
{  
    private Set theSet;  
  
    public void setTheSet(Set theSet)  
    {  
        this.theSet = theSet;  
    }  
}
```

Spring.xml

<beans>

```
<bean id="a" class="A">  
    <property name="theSet">  
        <set>  
            <value>100</value>  
            <value>Sathyu</value>  
            <value>100</value>  
        </set>  
    </property>  
</bean>
```

```
<bean id="b" class="B"/>
```

<beans>

→ internally spring container will do the following work for injecting the dependency.

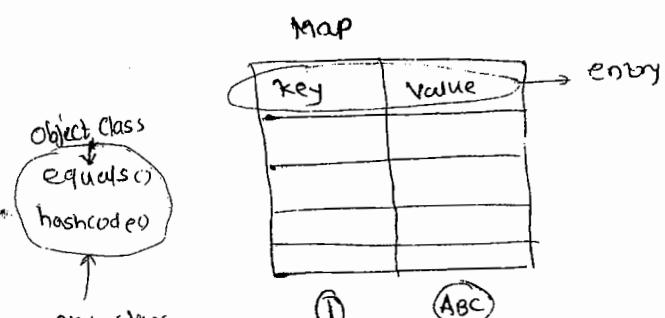
```
A a = new A();  
Set theSet = new LinkedHashSet();  
theSet.add(100);  
theSet.add("Sathyu");  
B b = new B(); ← 100 omitted as repeated  
theSet.add(b);  
a.setTheSet(theSet);
```

24-2-2015

Map tag

Map
HashMap
LinkedHashMap

Hashtable → Hashtable



which override this method are allowed for Key in map

check if this string

"Aa"
"BB" → contain same hashCode
↑ e.g. 2112

✓ String s1 = "Aa";
String s2 = "BB".
System.out.println(s1.hashCode()); → 2112
System.out.println(s2.hashCode()); → 2112

Class A

```
{
    void m1();
}
```

A a = new A();

System.out.println(a.hashCode()); → int value because hashCode() does not override here.
→ memory address converted int int format)

Class A

```
{
    public int hashCode()
    {
        return 100;
    }
}
```

void m1()

}

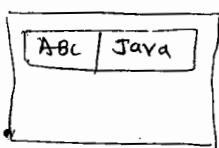
A a = new A();

System.out.println(a.hashCode()); → 100, because hashCode() method is overridden

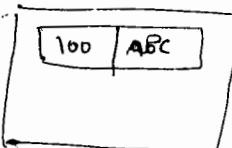
A a₁ = new A (10, 20);

A a₂ = new A (10, 20);

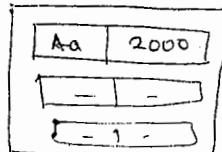
Bucket 1 (80)



Bucket 2 (100)



Bucket 3 (2112)



m.put (100, "ABC");

m.put ("Aa", 2000); → hashCode of Aa = 2112

m.put ("ABC", Java); → Suppose hashCode of ABC = 80

Object o = m.get ("Aa");

↓
hashCode() → finding right bucket if not found then create.
↓
equals() → finding right key

< Map > tag

→ We configured <map> tag in spring xml file for the following collections type.

- ① Map
- ② HashMap
- ③ LinkedHashMap
- ④ Hashtable

→ For Map, HashMap & LinkedHashMap collection type, the Spring IOC Container creates an object of LinkedHashMap

→ For Hashtable, container creates a Hashtable object

→ TreeMap is not supported

- in XML file under <Map> tag we configure <entry> tags
- in Map collection a ~~key-pair~~ key-value pair is called one entry.
- to use an object ~~as~~ as key, that class must override hashCode() & equals() methods
- internally a data of Map will be stored in hashCode buckets. So to store a ~~key~~ key/value pair in right bucket, hashCode() method on the key will be called.
- when reading the value from Map object based on the key, then first hashCode() methods call to find the right bucket, and then equals() method call to find the value in a the bucket.
- the possible combination of attributes in <entry> tags are
 - ① Key, Value
 - ② Key, value-ref.
 - ③ Key-ref, value.
 - ④ Key-ref, value-ref.

Ex:

```
public class A
{
    private Map theMap;
    public void setTheMap(Map theMap)
    {
        this.theMap = theMap;
    }
}
```

Spring.xml

```
<beans>
    <bean id="a" class="A">
        <property name="theMap">
            <map>
                <entry key="1" value="1000"/> → Key, Value
                <entry key="k1" value-ref="id3"/> → Key, Value-ref
            
    

```

```
<entry key-ref = "id1" value = "java" /> → key-ref, Value  
<entry key-ref = "id2" value-ref = "id4" /> → key-ref, value-ref
```

</map>

</property>

</bean>

```
<bean id = "id1" class = "X" />  
<bean id = "id2" class = "Y" />  
<bean id = "id3" class = "Z" />  
<bean id = "id4" class = "M" />
```

</beans>

internally IOC container will do the following

```
A a = new A();  
Map theMap = new LinkedHashMap();  
theMap.put(1, 1000);  
Z id3 = new Z();  
theMap.put("K1", id3);  
X id1 = new X();  
theMap.put(id1, "Java");  
Y id2 = new Y();  
M id4 = new M();  
theMap.put(id2, id4);  
a.setTheMap(theMap);
```

<props> tag

- if the dependency collection is Property class then in xml we configure <props> tag
- Property class object can store the data in String as key and String as value.
- other type of objects are not allowed.
- under <props> tag we configure a subelement <prop> tag

Ex:

```
public class A
{
    private Properties theProps;
    public void setTheProps(Properties theProps)
    {
        this.theProps = theProps;
    }
}
```

Spring.xml

```
<beans>
    <bean id="a" class="A">
        <property name="theProps">
            <props>
                <prop key="1">106</prop>
                <prop key="k1">Sathya</prop>
                <prop key="IS.67">abc</prop>
            </props>
        </property>
    </bean>
</beans>
```

Internally IOC Container will do the following.

```
A a = new A();  
Properties theProps = new Properties();  
theProps.setProperty("i", "100");  
theProps.setProperty("k1", "Sathyu");  
theProps.setProperty("15.67", "abc");  
a.setTheProps(theProps);
```

ApplicationContext Container

25-2-2015



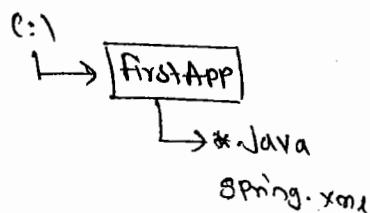
```
ApplicationContext ctx = new FileSystemXmlApplicationContext("D:\\Sathyu\\spring.xml");
```

ApplicationContext

- ApplicationContext is subinterface of BeanFactory
- the 3 important implementation classes of ~~interface~~ ApplicationContext interface are
 - ① ClassPathXmlApplicationContext
 - ② FileSystemXmlApplicationContext
 - ③ XmlWebApplicationContext → used in spring MVC application
- XmlWebApplicationContext will be used in spring MVC applications
- ClassPathXmlApplicationContext loads the spring configuration file from classpath location.
- classpath location means, either the XML file is present in the same location of class in which that XML file is loaded or it exist in

in jar file i.e added in classpath.

Example 1



C:\firstApp> javac Main <

→ Here Spring.xml is loaded in main class and both main class & Spring.xml are in the same directory. so xml file is in classpath location

Example 2

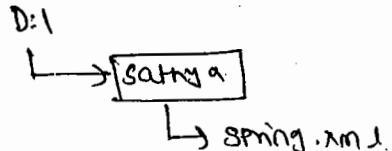


C:\ firstApp> set classpath = D:\a.jar ; <
C:\ firstApp> javac Main <

→ Here the xml file is in jar file & that jar is added to classpath so, the xml file is in classpath location.

→ if a class which is loading the xml file and the xml file are in two different location then we use FileSystemXmlApplicationContext for loading the xml file.

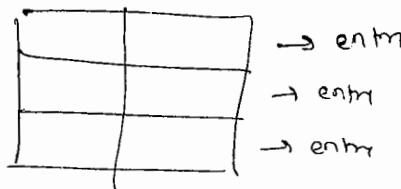
Example



```
ApplicationContext ctx = new FileSystemXmlApplicationContext("D:\Satya\Spring.xml");
```

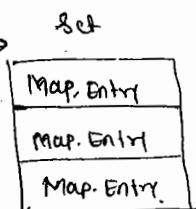
Map is best collection of Java

theMap



Set S = theMap.entrySet();

Iterator.

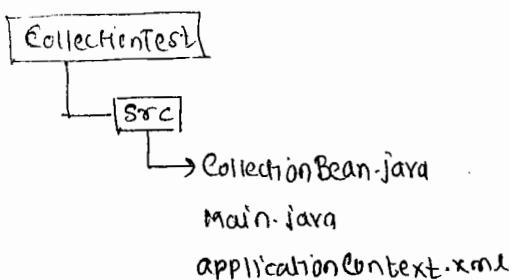


↓

Obj created

Map.Entry

(eclipse)



CollectionBean.java

```
import java.util.Iterator;  
import java.util.Map;           ctrl + shift + O → for import  
import java.util.Set;
```

```
public class CollectionBean
```

{

```
    private Set theSet;
```

```
    private Map theMap;
```

```
    public void setTheSet (Set theSet)
```

{

```
    this.theSet = theSet;
```

}

```
{ public void setTheMap(Map theMap)
```

```
{ this.theMap = theMap;
```

```
public void displayTheSet()
```

```
{ Iterator it = theSet.iterator();
```

```
while (it.hasNext())
```

```
{ Object o = it.next();
```

```
System.out.println(o.toString());
```

```
public void displayTheMap()
```

```
{ Set s = theMap.entrySet();
```

```
Iterator it = s.iterator();
```

```
while (it.hasNext())
```

```
{ Map.Entry me = (Map.Entry) it.next();
```

```
System.out.println(me.getKey() + " : " + me.getValue());
```

- Configure the following jar following build file to the Build Path of the Project

Commons-logging-1.0.4

Spring-beans-4.1.2

Spring-context-4.1.2

Spring-context-support-4.1.2-jar

Spring-core-4.1.2

Spring-expression-4.1.2

ApplicationContext.xml

```
<beans>
```

```
  <bean id = "collectionBean" class = "CollectionBean">
```

```
    <property name = "theSet">
```

```
      <set>
```

```
        <value> 100 </value>
```

```
        <value> Sathyu </value>
```

```
        <value> 100 </value>
```

```
    </set>
```

```
    <property>
```

~~```
 <!-->
```~~

```
 <property name = "theMap">
```

```
 <map>
```

```
 <entry key = "z" value = "1000"/>
```

```
 <entry key = "k1" value = "Java"/>
```

```
 <entry key = "15.15" value = "206.45"/>
```

```
 </map>
```

```
 </property>
```

```
 </bean>
```

```
</beans>
```

// Main.java,

②

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

```
 -->
```

```
 .support. ClassPathXmlApplicationContext;
```

```
public class main
```

```
 {
```

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

```
}
```

```
 ApplicationContext ctx = new ClassPathXmlApplicationContext
 (" applicationContext.xml ");
```

```
Collection Object o = ctx.getBean (" collectionBean ");
```

```
CollectionBean cb = (CollectionBean) o ;
```

```
System.out.println (" set values : ");
```

```
cb.displayTheSet();
System.out.println("Map Values :");
cb.displayTheMap();
}
```

dp Set Values:

100  
Sathya  
Map Values

k1: 1000  
k2: java  
15.15: 200.45

## Circular Dependency Issue.

- If a class contains dependency as reference of another class & if it has a setter injection then spring IOC container first creates an object of caller Bean then creates the object of dependency Bean.
- If a constructor injection is defined then container first creates an object of dependency Bean & then creates an obj. of dep. caller Bean.
- If A & B are two Beans, where A depends on B, B depends on A and in the both Beans constructor injection is defined then circular dependency problem occurs.

dependency as reference	Caller Bean	Dependency Bean
Setter in constructor	1st obj	2nd obj
	2nd obj	1st obj

class A

{

private B b;

public A(B b)

{

this.b = b;

}

}

class B

{

private A a;

public B(A a)

{

this.a = a;

}

}

- For creating an object of class A, container is waiting for class B object and for creating B object, container is waiting for A. Finally container throws BeanInstantiationException.
- To resolve above circular dependency problem, either in caller class or in a dependent class, we need to change the dependency type from constructor to setter.
- For example if we change the dependency injection type in class A, from constructor to setter, then container will do the following

- ① Creates a Mock object of A
- ② Injects Mock object to B through constructor.
- ③ finally Injects object of B to A through setter.

```
Class A
{
 private B b;
 public void setB(B b)
 {
 this.b = b;
 }
}
```

```
Class B
{
 private A a;
 public B(A a)
 {
 this.a = a;
 }
}
```

### Differences Betw Constructor & Setter injections

#### Constructor

- ① with constructor injections we cannot solve circular dependency problem.
- ② if dependency is of final then we need constructor for injecting it.
- ③ if you want to use dependency in constructor also, then we choose constructor injection.
- ④ if a dependency is a mandatory then we define constructor injection.
- ⑤ In this case, constructor injection is highly recommended, as we can inject all dependencies with in 3 to 4 lines [i.e. mean calling one constructor]

#### Setter

- ① with a setter injection we can solve circular dependency problem.
- ② for the dependency of final types, we cannot define setter.
- ③ if we don't want to use dependency in a constructor then we choose setter method.
- ④ if a dependency is optional then we choose setter injection.
- ⑤ If we have more dependences for example 15 to 20 are there in our bean class then in this case setter injection is not recommended as we need to write almost 20 setter right, bean length will be increased.

V Imp

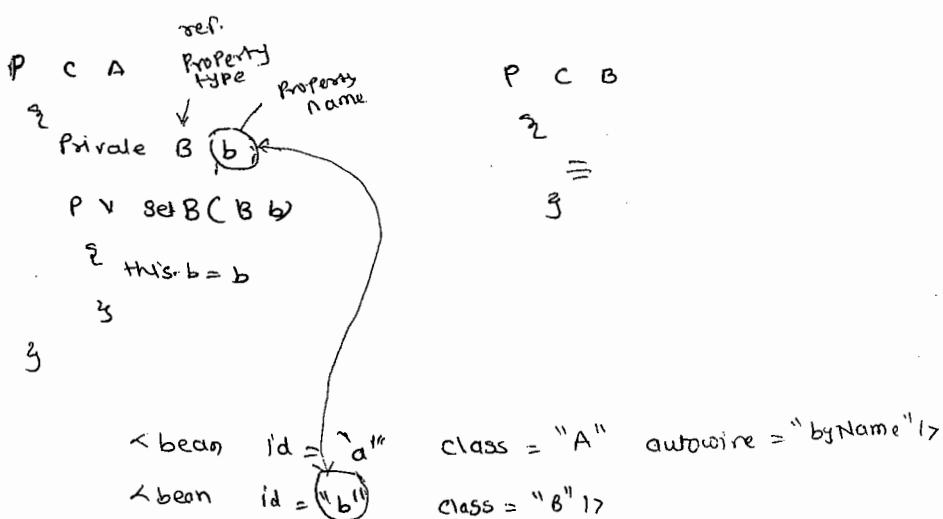
## Bean Autowiring

Bean autowiring

not applicable for primitive & collection type.

Applicable for ref. type.

28-2-2015



- the process of combining our bean classes with spring container is called bean wiring.
- for bean wiring, we need to configure our beans with there dependencies in an xme file.
- In bean autowiring, a spring container automatically inject the collaborator (dependencies) of the bean by without writing `<constructor-arg>` or `<property>` tags in Spring Configuration file.
- bydefault autowiring on bean is disabled, if we want to enable then we need to add autowire attribute in `<bean>` tag.
- the possible values of autowire attribute are
  - ① no (default value)
  - ② byName
  - ③ byType
  - ④ constructor

### ② byName

- if autowire strategy is a byName then a spring container verifies a bean id in xme is matched with property name or not. if matched then container injects an object of Bean class by calling the setter method.

for example

Bean id & Property name  
are checked.

P C A  
{  
    Private B b;  
    P v setB(B b);  
    { this.b = b;  
    }  
}  
g

P C B  
{  
    =;  
    }  
}

< bean id = "a" class = "A" autowire = "byName" />  
< bean id = "b" class = "B" />

→ if a bean id is not matched with a property name in xml then Spring container will not apply dependency injection. so that property remains with its default value "null".

→ if a bean id is matched with a property name, but its class is not matched then at the time of injecting the dependency java.lang.ClassCastException is thrown.

### ③ by Type

Bean class & Property type both are same

→ In this autowiring strategy, Spring container verifies a bean class in xml is matched with property type or not, if matched then container injects by calling setter method.

→

P C A  
{  
    Private B b;  
    P v setB(B b);  
    { this.b = b;  
    }  
}  
g

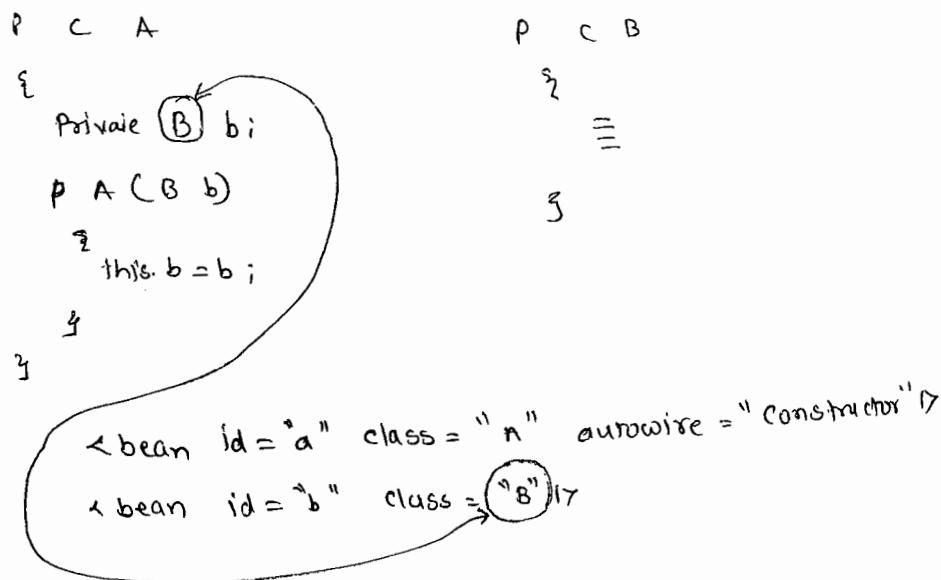
P C B  
{  
    =;  
    }  
}

< bean id = "a" class = "A" autowire = "byType" />  
< bean id = "b" class = "B" />

- If a bean class in XML is matched with property type for more than one's, Spring container throws `org.springframework.beans.UnsatisfiedDependencyException`.
- If a bean class in XML is not matched with property type then container will not inject that dependency, so that property remains <sup>with</sup> default value as "null".

#### (4) Constructor

- If autowiring strategy is constructor then Spring container checks a bean class in XML is matched with property type or not. If matched then container injects the dependency by calling constructor of the bean.
- The autowiring strategy byType & constructor has same verification strategy, but dependency injection type is different.
- The autowiring strategy byName & byType has same injection type i.e. setter, but verification is different.



- If a bean class is matched than more than one's, then container throws Unsatisfied Dependency Exception.
- If bean class in XML is not matched with a property type, then container call search for default constructor, if exist then an object of caller class created by calling the default constructor.

→ If a default constructor is not present, spring container throws Exception.

Q In which Autowiring strategy unsatisfied Dependency Exception doesn't occurs?

→ byName

P c MovieLister  
{  
    Private MovieFinder finder;

Q What is a drawback of Autowiring?

→ autowiring is only applicable for reference type but not applicable for primitive types & collections type.

Initialising a bean.

## Initializing a Bean Class

→ Before an object of a Bean class is going to become into usable state, if we want to execute any initializing logic on it then framework has given 2 options for initializing

- ① We can implement our Bean class from InitializingBean interface.
- ② We can define a custom init method in Bean class.

→ If we implement Initializing interface then we need override its abstract method afterPropertiesSet, for defining initialization logic

### Example

P C CurdBean implements InitializingBean interface

{

Private Datasource ds;

⇒ setter method

#### ② Override

Public void afterPropertiesSet() method

{

// Connection open logic

}

P V insert()

{  
=

P V delete() }

P V update() }

P V select() }

}

- In above Spring container automatically calls afterPropertiesSet() method, after the dependency is injected
- in the above our Bean class implementing framework given interface, it means our Bean class is acting as invasive class
- If we want to make our class as non-invasive class, we can define a custom init() method in <sup>our</sup> Bean class

```

public class CurdBean
{
 private DataSource ds;
 ⇒ setter method.

 public void setUp () → custom init method.
 {
 // connection open logic
 }

 public void insert () { }
 public void update () { }
 public void delete () { }
 public void select () { }
}

```

- to tell the Spring container that a method of our Bean class as init() method, we need to configure init-method attribute with Bean tag in XML.

#### XML configuration:

```
<bean id = "CurdBean" class = "CurdBean" autowire = "byName" />
```

```
<bean id = "ds" class = "BasicDataSource" init-method = "setUp" />
```

```
<bean id = "ds" class = "BasicDataSource" />
```

## Disposing is bean class:

→ before an object of an bean class is going to be garbage collector, if we want to execute some destruction logic of the bean by IOC container then the framework has given 2 options for defining the destruction logic.

- ① by implementing our bean class <sup>from</sup> ~~as~~ DisposableBean interface.
- ② by defining a custom destroy method.

→ If we implement DisposableBean interface then we need to override its abstract method destroy() for defining the destruction logic.

P c CardBean implements InitializingBean, DisposableBean

§

Private DataSource ds;

⇒ setter method

③ override

Public void afterPropertiesSet()

§

// Conn. open logic

¶

④ override

Public void destroy()

§

// Conn. close

¶

||

¶

→ If we want to make our bean class non-invasive class, we can define a custom destroy method & we can configure that method in destroy-method attribute in bean tag.

Public class CurdBean

၁

```
private DataSource ds;
```

⇒ setter method

public void setup() → custom init method

۹

// Conn. open logic

3

public void tearDown() → custom destroy method

乞

// Connection close

3

一  
二  
三

乙

## XML Configuration.

```
<bean id = "curdBean" class = "CurdBean" autowire = "byName"
init-method = "setUp" destroy-method = "tearDown">
```

```
<bean id = "ds" class = "BasicDataSource"/>
```

## Interface Injection in Spring

Interfaces.

method

① BeanNameAware → setBeanName(<sup>Name</sup> (String id))

② BeanFactoryAware → setBeanFactory(Beanfactory factory)

③ ApplicationContextAware → setApplicationContext (ApplicationContext ctx)

④ this method is called by Container for injecting bean id dependency

→ If an interface given method is called for injecting a dependency into a bean class then it is called interface injection.

→ Interface injection is not possible for the user defined interfaces.

→ Interface injection can't be configured in XML file.

→ Interface injection is possible only in the following 3 cases

① when we want to inject a bean id into the bean class.

② when we want to inject a Beanfactory container object into the bean class

③ when we want to inject a ApplicationContext container object into the bean class

→ Spring framework has provided 3 Aware Interfaces of Interface injection

① BeanNameAware

② BeanFactoryAware

③ ApplicationContextAware

① If a bean wants its id configure in XML then we need to implement that class from BeanNameAware interface.

→ the Spring Container calls setBeanName Method of BeanNameAware interface, for injecting a bean id.

for example → Next Page

2-3-2015

P C MyBean implements BeanNameAware

{  
    private String id;

@Override

    public void setBeanName(String id)  
    {  
        this.id = id;  
    }

}

- ② → if a bean wants a Beanfactory object then we need to implement our bean class from BeanfactoryAware interface.

P C MyBean implements BeanfactoryAware

{  
    private Beanfactory factory;

@Override

    public void setBeanfactory(Beanfactory factory)  
    {  
        this.factory = factory;  
    }

}

3

- setBeanfactory() method of BeanfactoryAware interface will be called by Spring Container for injecting a Beanfactory object.

- ③ → If a bean want ApplicationContext Object then we need to implement that bean class from ApplicationContextAware interface

P C MyBean implements ApplicationContextAware

{  
    private ApplicationContext ctx;

@Override

```
public void setApplicationContext (ApplicationContext ctx)
```

```
{
 this.ctx = ctx;
}
```

```
}
```

```
3
```

- setApplicationContext method of ApplicationContextAware interface called by container for injecting ApplicationContext object.

Scope → lifetime of object

### \* Bean Scopes:

- a scope indicates lifetime of an object or value within a container.
- in order to set a scope for bean, we need to configure scope attribute along with <bean> tag
- if we do not configure scope attribute then by default Spring container reads the scope as singleton.
- the available scopes for bean in spring are
  - ① singleton
  - ② prototype
  - ③ request } MVC
  - ④ session.
  - ⑤ global session. → Portlet
- global session scope can be used Spring with portlet.
- request and session scope can be set for a bean only in MVC applications.
- if we set bean scope as a singleton then Spring container will creates one object of that bean class with respect to the Id and returns same object for multiple times.

### for example ①

```
<bean id = "id1" class = "A" scope = "singleton" />
```

```
A a1 = (A) ctx.getBean("id1");
```

```
A a2 = (A) ctx.getBean("id1");
```

$a_1 == a_2 \Rightarrow \text{true}$

for one id one  
object in singleton

i.e for id1 it will  
create one object

& for id2 it will create  
one object

### ② for example

```
<bean id = "id1" class = "A" scope = "singleton" />
```

```
<bean id = "id2" class = "A" scope = "singleton" />
```

```
A a1 = (A) ctx.getBean("id1");
```

```
A a2 = (A) ctx.getBean("id2");
```

```
A a3 = (A) ctx.getBean("id1");
```

```
A a4 = (A) ctx.getBean("id2");
```

$a_1 == a_2 \Rightarrow \text{false}$

$a_3 == a_4 \Rightarrow \text{false}$

$a_1 == a_3 \Rightarrow \text{true}$

$a_2 == a_4 \Rightarrow \text{true}$

**Note** → Spring Container make <sup>bean</sup> object as singleton w.r.t to id.  
It means when we call multiple times getBean() method by  
passing the same id, the container returns same object of  
the bean.

→ If the bean scope is prototype then Spring Container creates &  
returns a new object whenever getBean() method is called

```
<bean id="id1" class="A" scope="prototype">
<bean id="id2" class="A" scope="prototype">
```

A a<sub>1</sub> = (A) ctx.getBean("id1");

A a<sub>2</sub> = (A) ctx.getBean("id1");

a<sub>1</sub> == a<sub>2</sub> → false

A a<sub>3</sub> = (A) ctx.getBean("id2");

a<sub>1</sub> == a<sub>3</sub> → false

a<sub>2</sub> == a<sub>3</sub> → false

### Making a Java class as Singleton:

- if we created a Java class in ~~one~~ an application with fixed data  
and ~~if~~ multiple objects are created to that the java class then  
memory for the objects will be wasted.
- to solve this memory problem, one object of the java class should be  
created and it should be made ~~be~~ sharable to all other classes.
- To make a java class as 100% Singleton class we need to follow below  
rules.

- ① We should not allow other java classes to create an  
object of a singleton class. to restrict the, we need to  
make constructor of that class as private
- ② we need to create a ~~so~~ global access point (factory access method)  
to read a object of singleton class by other classes.

③ In multithreading application, if two threads calls the factory method concurrently or simultaneously then two object of the singleton class will created. To restrict two thread at a time we need to make a factory method as synchronized method.

④ A singleton class object should not be cloned, to avoid cloning we need to override clone method and throw ~~classe~~ CloneNotSupportedException

Public class A implements Cloneable

{  
① Private A () { } }

② Private static A a;

③ ④ Public static synchronized A getInstance()

{  
if (a == null)

a = new A ();

}

return a;

}

⑤ Public Object ~~clone()~~ clone () throws CloneNotSupportedException

{  
throw new CloneNotSupportedException();

}

→ In a spring configuration file, we need to configure our singleton class like the following.

```
<bean id = "id1" class = "A" factory-method = "getInstance" scope =
"prototype"/>
```

→ even though scope is prototype but the bean class is always singleton, because spring container also calls factory-method only to read the object.

A a1 = (A) cxt.getBean("id1");

A a2 = (A) cxt.getBean("id1");

a1 == a2 → true.

Look-up injection

4-9-2015

### Look-up Method injection

- ① Suppose in an application, we have two bean classes A & B where A depends on B
- ② A is a Singleton bean and B is a Prototype Bean.
- ③ In class A, we have taken reference variable of class B as dependency and we defined setter or constructor for injecting dependency.

#### Example

```
public class A
{
 private B b;
 public void setB(B b)
 {
 this.b = b;
 }
}
```

```
public class B
{
 public void m2()
 {
 }
}
```

## XML configuration

```
<bean id = "id1" class = "A" scope = "singleton">
```

```
 <property name = "b" ref = "id2" />
```

```
</bean>
```

```
<bean id = "id2" class = "B" scope = "prototype" />
```

→ In the above, even though the scope of class B is Prototype but Spring container injects one object of class B only to class A. because dependency of a class are injected for once, whenever its object is created.

→ if we want to inject multiple objects of prototype bean to one object of singleton bean then instead of constructor or setter injection, we need to apply lookup method injection.

→ for injecting a multiple objects of prototype bean to one object of singleton bean using lookup method injection we need to do the following.

- ① we need to create an abstract method in class A with return type as class B
- ② we need to make class A as a abstract because of abstract method
- ③ In methods of class A we need to call abstract method to get an object of class B

```
abstract public class A
```

```
{
```

```
 public abstract B getObjectOfB();
```

```
 public void m1()
```

```
{
```

```
 getObjectOfB().m2();
```

```
}
```

```
g
```

- at runtime Spring container creates a proxy class by extending class A & in that proxy class it overrides abstract method and returns a new object of class B.
- In the above if `get()` method of class is called two times or twice then `getObjectOfB()` method returns an object of class B for twice.
- in XML configuration file we need to configure the abstract method of class A as look-up method like the following.

```

<bean id = "id1" class = "A" scope = "singleton">
 <lookup-method name = "getObjectOfB" bean = "id2"/>
</bean>
<bean id = "id2" class = "B" scope = "prototype"/>

```

- Singleton bean is depend on Prototype bean

- Q How can we inject an object created using Proxy design pattern into object created using Singleton design pattern.
- <sup>using</sup> lookup-method injection.

## P-namespace, C-namespace, & util-namespace.

- these namespaces are included by spring in 2.5 version.
- P-namespace is given to avoid to remove <Property> tags from an xml file.
- C namespace is given to avoid to remove <constructor-arg> tags from xml file.
- Util namespace is given to configure a collection for xml separately to make it as reusable for multiple beans.
- by using above 3 namespaces a lot of xml is reduced.
- if we want to use ~~this~~ those namespaces then we need to import the namespaces into a xml file like the following.

```

xmlns:p = "http://www.springframework.org/schema/p"
xmlns:c = "http://www.springframework.org/schema/c"
xmlns:util = "http://www.springframework.org/schema/util"

```

Ex:

public class A

```

{
 private int x; → dependency is primitive type
 private B b; → reference type / object type
 private List theList; → collection type

```

public A (int x)

```

{
 this.x = x;
}

```

public void setB (B b)

```

{
 this.b = b;
}

```

```

 public void setTheList(List thelist)
 {
 this.thelist = thelist;
 }
}

```

### XMLE Configuration without using namespace

```

<bean id="id1" class="A">
 <constructor-arg value="100"/>

<!-- bean id="id2" class="A" -->
 <property name="b" ref="id2" />
 <!-- id="id3" class="A" -->
 <property name="thelist">
 <list>
 <value>10</value>
 <value>20</value>
 </list>
 </property>
<bean id="id2" class="B">

```

### XMLE Configuration with namespace

```

<bean id="id1" class="A" c:k="100"
 p:b-ref="id2" p:thelist-ref="li"/>

```

```

<bean id="id2" class="B"/>
<util:list id="li">
 <value>10</value>
 <value>20</value>
</util:list>

```

→ the following are the 4 tags of util namespace.

- ① <util:list>
- ② <util:set>
- ③ <util:map>
- ④ <util:Properties>

→ if we configure util:list tag then by default the Spring container creates an object of ArrayList (java.util.ArrayList).

→ If we want to customize class name then we need to configure list-class attribute

```
<util:list id="li">
 list-class = "java.util.LinkedList">

</util:list>
```

→ If we configure <util:set> then container creates by default an object of LinkedHashSet, if we want to customize then we need to add set-class attribute

```
<util:set id="s" set-class = "java.util.TreeSet">
 -- -- --
</util:set>
```

→ If we configure <util:map> then by default Container creates an object of LinkedHashMap, if we want to customize then we need to add map-class attribute

```
<util:map id="m" map-class = "java.util.TreeMap">
 -- -- --
</util:map>
```

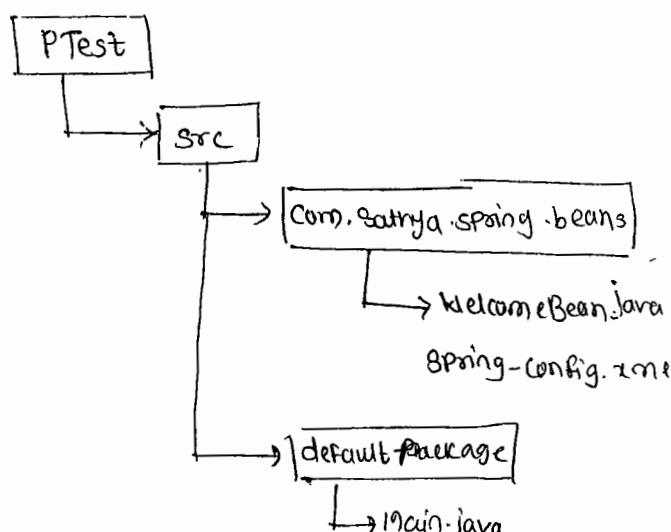
→ for <util:properties> tags, the container creates a Properties class object  
Properties is already a class only so we cannot customize class again.

```
<util:properties id="p">
 <prop key="d">1000</prop>
 <prop key="a">satya</prop>
</util:properties>
```

```
Map m = new HashMap HashMap();
m.put("a", "100");
m.put("b", "200");
System.out.println(m); → {a=100, b=200}
```

for Set & List Obj in [ ]  
for Map (key,value) Obj in { }  
for Map (key,value) Obj in { }

6-3-2015



## // WelcomeBean.java

```
Package com.Satya.Spring.beans;
import java.util.Map;
public class WelcomeBean
{
 private String message;
 private Map theMap;

 public void setTheMap(Map theMap)
 {
 this.theMap = theMap;
 }

 public void setMessage(String message)
 {
 this.message = message;
 }

 public void showMessage()
 {
 System.out.println(message);
 }

 public void showTheMap()
 {
 System.out.println(theMap.getClass().getName());
 System.out.println(theMap);
 }
}
```

← -- Spring-config.xml --→

```
<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/beans
 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
 http://www.springframework.org/schema/util
 http://www.springframework.org/schema/util/spring-util-4.0.xsd">
```

```
<bean id="welcomeBean" class="com.sathyaj.Spring.beans.WelcomeBean">
 P: message = "welcome to spring" ↴
<util:map id="m1" map-class="java.util.HashMap">
 P: theMap-ref = "m1" ↴
 <entry key="j" value="java"/>
 <entry key="s" value="100"/>
 </util:map>
</beans>
```

```
// Main
class Main
{
 main {
 ApplicationContext ctx = new ClassPathXmlApplicationContext("spring-config.xml");
 Object o = ctx.getBean("welcomeBean");
 WelcomeBean welcomeBean = (WelcomeBean)o;
 welcomeBean.showMessage();
 System.out.println("=====");
 welcomeBean.showTheMap();
 }
}
```

/ → File  
\* → Java class

*"com/sathyaj/Spring/beans/spring-config.xml"*  
*write fully qualified name*

OP  
welcome to spring  
=====

java.util.HashMap

{ s=100, j=java }

## BeanPostProcessor interface.

Common bean initializing logic

P C MyBeanPostProcessor implements BeanPostProcessor

{  
    ② override

P Object postProcessBeforeInitialization (Object bean, String name)  
{  
    "  
}  
    3

② override  
P Object postProcessAfterInitialization (Object bean, String name)  
{  
    "  
}  
    3  
}

- In a Spring Application, if a multiple beans are having common initialization logic and if it is define in each bean separately then duplication of the code occurs in a application.
- In order to avoid duplication of initialization logic, framework has given BeanPostProcessor interface.
- instead of duplicating common initialization logic we need to create a separate class by implementing BeanPostProcessor interface to define a common initialization logic.
- for example

Public class MyBeanPostProcessor implements BeanPostProcessor  
{

## @Override

```
public Object postProcessBeforeInitialization (Object bean, String name)
{
 ==
}
```

## @Override

```
public Object postProcessAfterInitialization (Object bean, String name)
{
 ==
}
}
```

→ As Part initialization of Spring bean Container 1st calls postProcess  
postProcessBeforeInitialization method, After that initialization method defined in  
the Bean class will be called. And finally postProcessAfterInitialization method  
will <sup>be</sup> called.

## for example

Public Class A

```
{
 public void setUp()
}{
 ==
}
```

[order of initialization]

- ① postProcessBeforeInitialization()
- ② setUp()
- ③ postProcessAfterInitialization()

Public Class B

```
{
 public void init()
}{
 ==
}
```

[Order of Initialization.]

- ① postProcessBeforeInitialization
- ② init()
- ③ postProcessAfterInitialization

# Bean Life Cycle

7-3-2015

- life cycle is for object.
- object is created by JVM by new keyword.
- JVM calls constructor

→ If we initialize object in method, we have to call that method.

Constructor  
→ ^ to indicate a JVM has finished object creation.

P C A

{ int x, y

void init()

{ x=10;

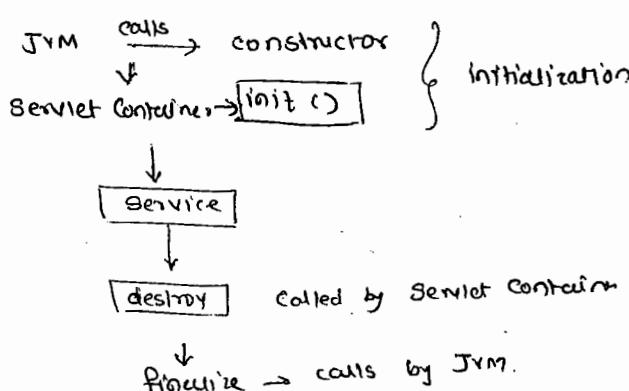
y = 20;

}

y

JVM  
↓  
create object  
↓  
calling constructor  
for initialization

Servlet → class      Servlet Container internally calls JVM to create object



⇒ **Ques**

→ life cycle of a Java class contains 5 well defined stages.

- ① does not exist
- ② instantiation.
- ③ initialization.
- ④ service.
- ⑤ destroy

→ Actually life cycle exist for object, not for a class. but we call object life cycle as class life cycle.

→ for every class in Java that is for POJO class, Servlet class, or Spring bean class etc. there is a life cycle.

→ **POJO life cycle!**

```
public class A
{
 void m1()
 void m2()
}
```

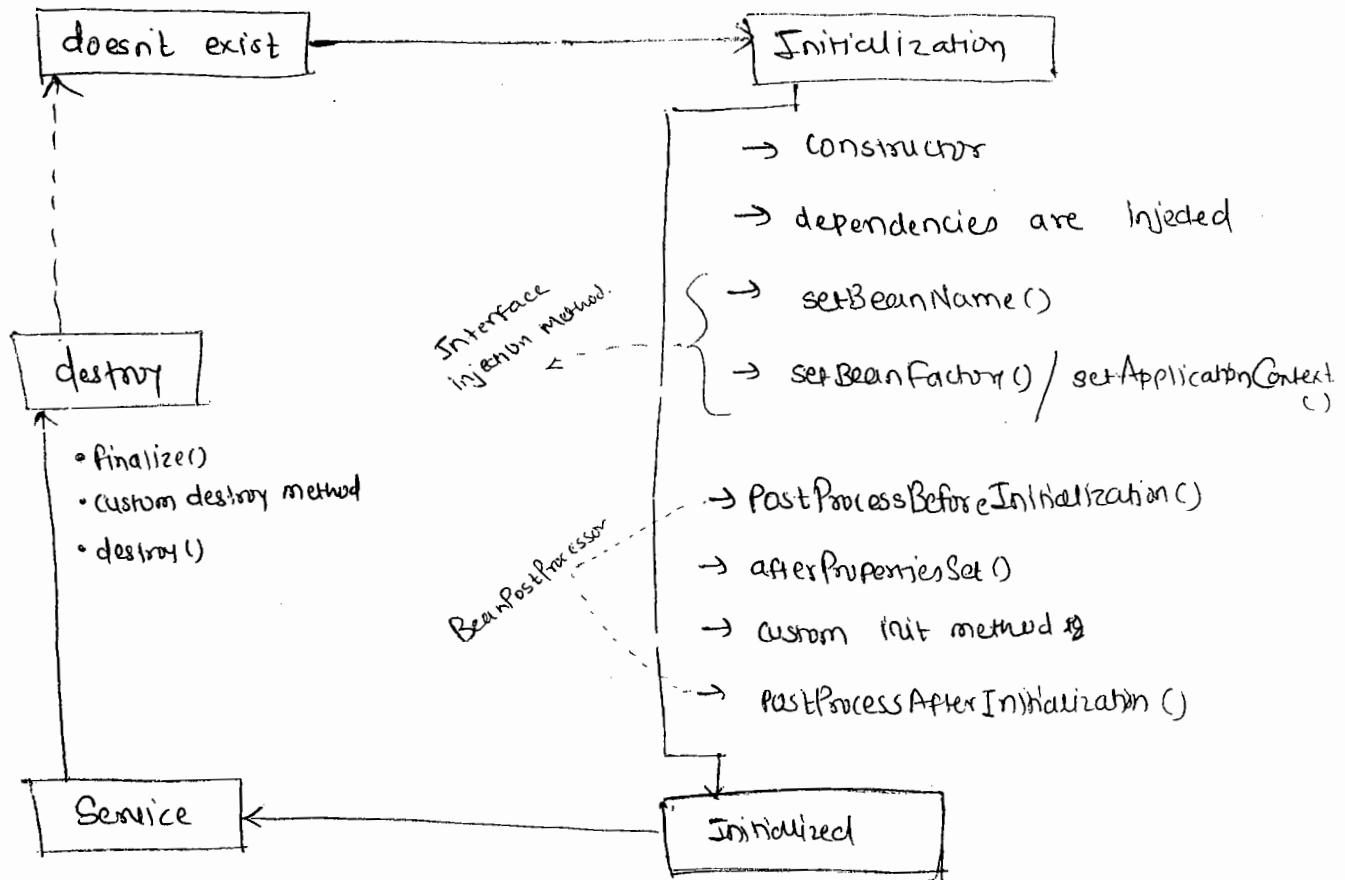
- ① before an object of class A is created, its life cycle stage is does not exist.
- ② ~~for~~ each Java class will be instantiated by JVM only.
- ③ after instantiation (memory allocation) JVM replies that instantiation is completed by calling the constructor.
- ④ if we defined any initialization logic in constructor then it will be executed. if initialization logic defined in any other method then that method is need to be called to finish initialization.
- ⑤ after initialization that object will provide the service, by calling the methods.
- ⑥ finally, before "object is going to garbage collector finalize()" method will be called by JVM, to indicate object is destroyed.

- does not exist
- A a = new A();
- Instantiated & initialized
- a.m1();
- a.m2();
- Service
- finalize()
- destroy

### Servlet life Cycle:

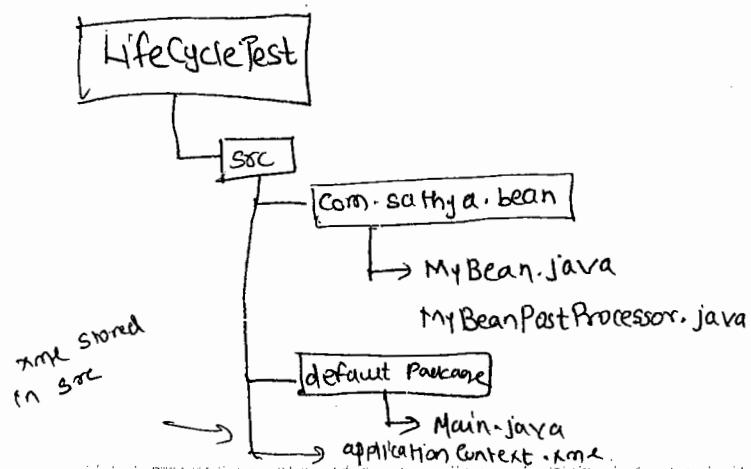
- In Servlet we say that life cycle will be managed by Servlet container.
- Servlet container also internally calls JVM only for instantiating a Servlet nothing but creating object.
- JVM replies servlet container that a servlet is instantiated by calling constructor. After that a servlet container calls init() method to finish initialization.
- After initialization service() method is called for each request to provide service to the clients.
- A Servlet container releases object to JVM by calling destroy() method and later JVM releases to garbage collector by calling finalize() method.
- So a Servlet object is completely destroyed.
- In the life cycle of the servlet also we have the same 5 stages

## Bean Life cycle



- In Spring framework, we say that, Spring Container creates a Bean Object but Spring container internally calls JVM only for creating an object.
- after instantiating bean class by JVM, the JVM calls constructor of the Bean class, so that initialization begins.
- after constructor is executed, the Spring container injects the dependencies required to Bean object.
- the Spring Container checks whether a Bean class implements BeanNameAware Interface or not, if yes then call setBeanName() method.

- a Spring Container checks a bean class implements BeanFactoryAware or ApplicationContextAware interface ~~or~~ or not.
- If BeanFactoryAware interface is implemented then calls setBeanFactory() method & if ApplicationContextAware interface is implemented then calls setApplicationContext() method.
- If BeanPostProcessor class is created then container calls postProcessBeforeInitialization() method.
- If a bean class implements InitializingBean interface then container calls afterPropertiesSet() method.
- If a custom init method is defined then it ~~will~~ will be called.
- postProcessAfterInitialization() of a BeanPostProcessor class is called.
- Now the bean is initialized and ready to provide to service.
- Before removing a bean object from container, it checks whether a bean class implements DisposableBean interface or not. if yes then it calls destroy() method.
- If a custom destroy method is defined then it will be called.
- Before a bean object is going to be garbage collected, jvm calls finalize() method. now the bean object is completely destroyed.



ApplicationContextAware

Public class MyBean implements BeanNameAware, ~~ApplicationContext~~, InitializingBean,  
DisposableBean

```
{
 Private String msg;

 Public MyBean ()
 {
 System.out.println ("constructor called");
 }

 Public void setMsg (String msg)
 {
 this.msg = msg;
 System.out.println ("dependency injected.");
 }
}
```

② Override

```
Public void destroy () throws Exception
{
 System.out.println ("destroy method called");
}
```

③ Override

```
Public void afterPropertiesSet () throws Exception
{
 System.out.println ("afterPropertiesSet method called.");
}
```

④ Override

```
Public void setApplicationContext (ApplicationContext arg0) zero
throws BeansException
{
 System.out.println ("setApplicationContext method called");
}
```

⑤ Override

```
Public void setBeanName (String arg0)
{
 System.out.println ("setBeanName method called.");
}
```

```
public void setup()
{
 System.out.println("custom init method called");
}

public void tearDown()
{
 System.out.println("custom destroy method called.");
}

public void mt()
{
 System.out.println("mt method called");
}
```

### (( MyBeanPostProcessor.java

```
public class MyBeanPostProcessor implements BeanPostProcessor
```

```
{
```

@Override

```
public Object postProcessAfterInitialization (Object arg0, String arg1)
throws BeansException {
 System.out.println("postProcessAfterInitialization method called");
 return arg0;
}
```

@Override

```
public Object postProcessBeforeInitialization (Object arg0, String arg1)
throws BeansException
{
 System.out.println("postProcessBeforeInitialization method called");
 return arg0;
}
```

## applicationContext.xml

<beans>

```

<bean id = "MyBean" class = "com.sathya.bean.MyBean" init-method = "setUp"
 destroy-method = "tearDown">
 <property name = "msg" value = "sathya" />

```

</beans>

<bean class = "com.sathya.bean.MyBeanPostProcessor" />

</beans>

// Main.java

public class Main {

Main() {

```

 ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext
 ("applicationContext.xml");

```

MyBean myBean = (MyBean) ctx.getBean("MyBean");

myBean.init();

--> ctx.close();

}

}

ctx.close() → method is not in ApplicationContext interface, that's why  
 → ClassPathXmlApplicationContext is taken.

o/p

Constructor called

Dependency injected.

SetBeanName method called.

SetApplicationContext method called.

PostProcessBeforeInitialization method called

afterPropertiesSet method called

Custom init method called.

PostProcessAfterInitialization method called

init method

destroy method called

Custom destroy method called

✓

## Core module annotations.

xml → Meta information.

xml has extensibility, flexibility.

- { ① Stereotype annotations
- ② autowiring annotations
- ③ miscellaneous annotations

- When working with Java technologies & Java frameworks, a programmer is need to write some configuration data in xml file.
- When size of the project is increasing, the size of xml files are also going to be increase.
- To cutdown the size of xml files, annotations are introduced in Java.
- With the help of annotations, we can also define meta data in source code only. So there is no need to define metadata in xml file separately.
- In Spring framework the framework has given annotations for each module to reduce the size of the spring configuration file.
- The annotations of a core module are divided into 3 types.

- ① Stereotype annotations.
- ② autowiring annotations
- ③ miscellaneous annotations

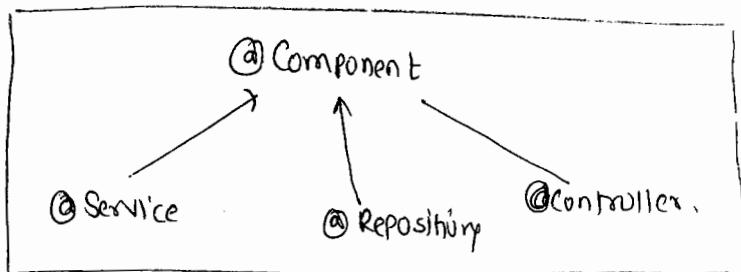
### Stereotype Annotations / ①

- Stereotype Annotations tells about role of a class in an application.
- A programmer can identified whether a class containing or consisting either business logic or persistence logic or controller logic easily with the help of stereotype annotation.

→ the framework has given 4 Stereotype annotations.

- { ① `@Component` → tells role of class
- ② `@Service` → business logic
- ③ `@Repository` → persistence logic
- ④ `@Controller` → controller bean of MVC application

→ `@Component` annotation is the base for the remaining 3 annotations



→ If we add `@Component` annotation on the top of class then it tells the a class is a component class. it means it is class consisting of some logic.

→ `@Component` annotations doesn't tell whether a class is specifically containing business or persistence or controller logic. So we don't use directly this `@Component` annotation.

→ if we add `@Service` annotation then it tells the class consisting business logic.

→ if we add `@Repository` annotation on top of class then it tells the a class consisting persistence logic.

→ if we add `@Controller` then it tells that a class is acting as Controller bean of MVC architecture application.

for example

for ex:-

Package com.satyaj.beans;

@Service

```
public class TestBean
{
 public void m1()
 {
 // Business code
 }
}
```

Package com.satyaj.beans;

@Repository

```
public class CrdBean
{
 public void insert()
 {
 // Persistence logic
 }
}
```

- the classes which are added with Stereo type annotations are going to be ~~visible~~ visible to spring container in a Component-scanning.
- Component scanning is a feature added in spring 2.5 where ~~where~~ a spring container will automatically configure the beans at runtime by scanning a given package.
- the benefit of component scanning is we no need to explicitly configure the beans in xml file so the size of the xml will be reduced.
- In component scanning a container identifies a class or package

as a bean, if a stereotype annotation is added on top of the class.

→ To enable component scan feature, we need to configure  
<context:component-scan> tag with package name in XML file

for example

<context:component-scan base-package = "com.sathya.beans"/>

→ the Spring container scans com.sathya.beans package & also its subpackages for the components.

→ the container configures a component as bean at runtime with unqualified class name as id, by starting it with a lower case letter

for example the id configured for com.sathya.beans.TestBean is testbean

②

## Autowiring Annotations

10-3-2015

→ bean Autowiring is a ~~feature~~ facility provided by Spring container, for automatically injecting dependencies or reference types, by without explicitly configuring it id

→ autowiring facility is only applicable for dependencies of reference types.

→ Annotation for Autowiring.

① @Autowired

② @Qualifier

- these autowired annotation is applicable for ~~beans~~ fields (variables), methods, & for constructors.
- Stereotype Annotation are class level annotations, it means they are only applicable for classes.
- if we add @Autowired annotation then a spring container checks for a bean class in xml is matched with property type or not. If matched then injects a that bean object by calling either setter or a constructor.
- if not matched then throws <sup>an</sup> exception.

### Example

// Movielister.java

package com.satyaj.beans;

@Service

public class Movielister

{

    @Autowired

    private Moviefinder finder;

    public void setfinder(Moviefinder finder)

{

        this.finder = finder;

}

}

// Moviefinder.java

package com.satyaj.beans;

@Service

public class Moviefinder

{}

Xml Configuration

<Context: component-scan base-package = "com.satyaj.beans" />

→ In the above example @ at the time of component scanning, the container finds a ~~new~~ Moviefinder which is matched with the property type in MovieLister, so it injects the an object of Moviefinder bean by calling setter method.

### Required attribute

→ If we want to tell the container that if bean class in XML is not matched with property type then don't throw an exception then we need to add required element to the @Autowired annotation.

public class MovieLister

{

    @.Autowired (required = false) [default value is true]

    private Moviefinder finder;

---

}

ie it throws the exception.

→ by passing required = false we are telling a spring container that a dependency is optional.

→ if constructor injection is defined then, even though required = false, container throws an exception to ~~overcome~~ overcome it we need to defined default constructor also

public class MovieLister

{

    @.Autowired (required = false)

    private Moviefinder finder;

    public MovieLister() [default constructor]

{}

    public MovieLister (Moviefinder finder)

{

        this. finder = finder;

}

}

## @Qualifier

- if a bean class in a xme is matched ~~with more~~ <sup>for more</sup> than one with property type, then ambiguity occurs and container throws an exception. to resolve it, we will use @Qualifier Annotation.
- @Annotation @Qualifier Annotation always used with combination with @Autowired annotation, it cannot be used independently

public class MovieLister

{

    @.Autowired

    @Qualifier("id2")

    private MovieFinder finder;

==

}

## XMLE Configuration

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

```
<bean id = "id2" class = "com.sathya.beans.MovieFinder"/>
```

③

### Miscellaneous annotations:

① @Scope:

→ this annotation is used to set the scope for a bean class

@Service

@Scope (value = "prototype")

public class A

{

}

② & ③) @PostConstruct & @PreDestroy

→ these annotations are used to tell the container about custom init method  
a custom constructor and a custom destroy method of bean class to the container

@Service

public class A

{

    @PostConstruct

        public void setup()

{

            // initialization logic

}

    @PreDestroy

        public void tearDown()

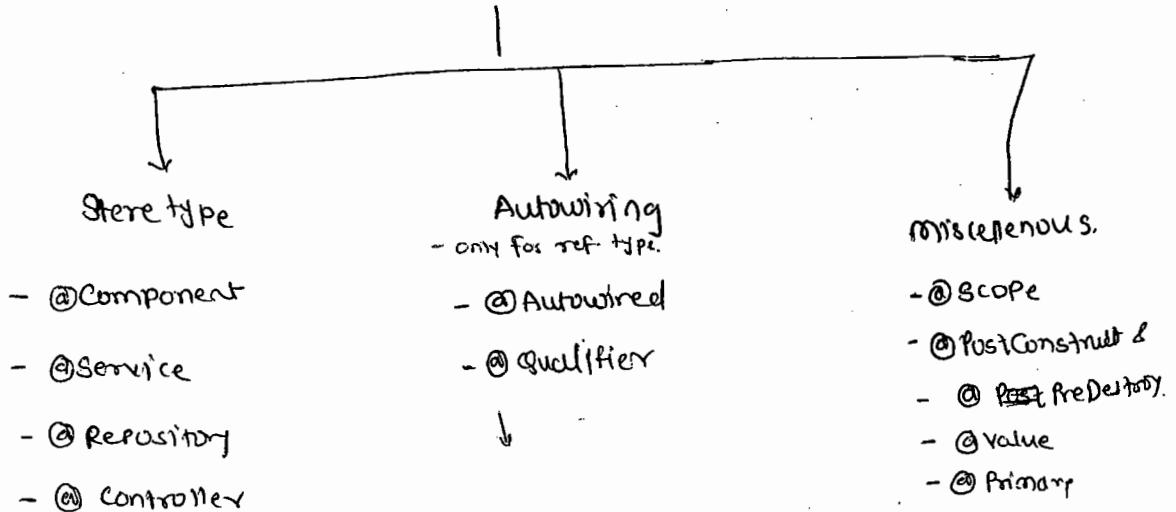
{

            // destruction logic

}

}

## Core module Annotation



#### ④ @Value:

→ this annotation is given for injecting a value to a primitive type or simple type property, by reading it from a resource container.

→ To configure a the location of the resource ~~model~~ model bundle, we need to configure <context:property-placeholder> tag in spring configuration file

public class A

```
{ @Value ("${xyz}") }
```

private int x;

=

}

Sathya.properties

```
R1 = 100
```

#### XML Configuration

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

#### ⑤ @Primary:

→ when multiple classes are implementing a common interface then, to tell the container give more priority to one implementation class we use @Primary Annotation on top of the class

package pack1;

@Service

P c Travel

{

@Autowired

private Vehicle v;

⇒ better

3

Package pack1;

@Service

→ @Primary

P c car implements  
Vehicle

{

5

Package pack1;

@Service

P c Bike implements

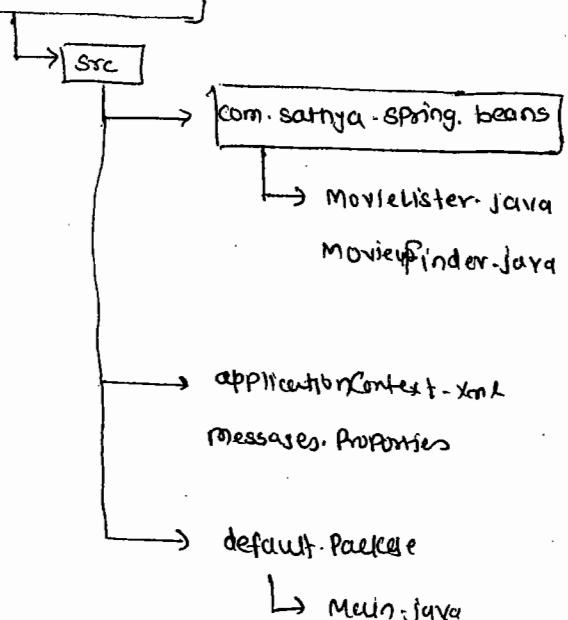
Vehicle

{

3

## Program

### Annotation Example



① // MovieLister.java

```
Package com.sathya.spring.beans;
import javax.annotation.PostConstruct;
@Service
@Scope ("prototype")
Public class MovieLister
{
 @Value ("${msg}")
 Private String message;
 @Autowired
 Private MovieFinder finder;
 Public void setMovieFinder (MovieFinder movieFinder)
 {
 this.finder = movieFinder;
 }
}
```

### ② PostConstruct

```
public void setup()
{
 System.out.println("I am setup method");
}
```

### ③ PreDestroy

```
public void tearDown()
{
 System.out.println("I am tearDown method");
}
```

```
public void showMessage()
```

```
{
 System.out.println(message);
}
```

```
public void m1()
```

```
{
 System.out.println("I am m1 of MovieLister");
 finder.m2();
}
```

② // Moviefinder.java

### ④ Component

```
public class Moviefinder
{
 public void m2()
 {
 System.out.println("I am m2 of MovieLister");
 }
}
```

### ③ application context.xml

<beans>

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

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

<context:component-scan base-package = "com.sathyaspring.beans"/>

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

<context:annotation-config />



this tag tells the container execute autowiring & miscellaneous annotation of core module.

### ④ #messages.properties

msg = Hello

### ⑤ Main.java

public class Main

{

    main()

        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");

        MovieLister lister = (MovieLister) ctx.getBean("MovieLister");

        lister.showMessage();

        lister.m1();

        ctx.close();

}

3

## List of jar

- ① aspelliance-1.0.jar → not from Spring software
- ② commons-logging-1.0.4.jar
- ③ spring-aop-4.0.5.RELEASE
- ④ spring-beans-4.0.5.RELEASE
- ⑤ spring-context-4.0.5.RELEASE
- ⑥ spring-context-support
- ⑦ spring-core
- ⑧ spring-expression

# Spring Expression Language (SpEL)

12-3-2015

$\$\{ \text{expression} \}$  → JSP

$\#\{ \text{expression} \}$  → Spring

name = "K" value =  $\#\{ \text{id}, K * 10^3 \}$

↑  
value assigned at runtime & dependent on K

→ 0

→ So far, we know that we can pass the static value to the properties of a bean at ~~at~~ compiled time in a XML file.

→ If we want to set a value to ~~the~~ a property, by reading it from another bean property or by calling method of another bean at run time. Then we use SpEL in Spring configuration file, we can pass expression as ~~value~~ value, so that it will be ~~a~~ evaluated at run time & its value will be injected to the properties at runtime.

## Example

```
f c A
{
 private int k;
 public void setK(int k)
 {
 this.k = k;
 }
 public int getK()
 {
 return k;
 }
}
```

```
p c B
{
 private int z;
 p v setZ(int z)
 {
 this.z = z;
 }
}
```

```

<bean id="id1" class="A">
 <property name="k" value="10"/>
</bean>

<bean id="id2" class="B">
 <property name="z" value="#{id1.k * 10}"/>
</bean>

```

- \* the syntax of Spring expression begins with #{} , end with }
- In the above expression it calls a getter method of property "A" at runtime, then multiplies with 10 & then injects that value to "z" property

### Example 2

P c Writer

```

 {
 private String songName;
 public void setSongName (String songName)
 {
 this.songName = songName;
 }
 }

```

P c Song

```

 {
 public String readSong()
 {
 return "xxxx";
 }
 }

```

## XML Configuration

```
<beans id="id1" class="Guitar">
 <Property name="songName"
 value="#{!id2.readSong()}" />
 </bean>
```

↳ calls at runtime.

```
<bean id="id2" class="Song" />
```

→ at runtime, a String value returned by ~~readSong~~ readsong() method will be set to songName property.

→ When calling a method on a method in a string expression, if first method returns null then by when calling the 2nd method NullPointerException will be thrown.

→ To avoid NullPointerException, SpEL has added a NullSafe Operator.  
null safe operator (?.)

→ the null safe operator checks whether a left side value is null or not null before calling the right side method. If left side value is null then it doesn't call right side method.

```
<bean id="id1" class="Guitar">
```

```
 <Property name="songName"
 value="#{!id2.readSong().?..toUpperCase()}" />
```

```
</bean>
```

```
<bean id="id2" class="Song" />
```

→ if we want to call either static method or static properties of a class in expression language then we use "T" operator

```
<bean id = "id1" class = "A">
 <Property name = "x"
 value = "# { T (java.lang.Math). Pow(2,3) }" />
```

```
</bean>
```

```
<bean id = "id2" class = "B">
 <Property name = "z"
 value = "# { T (java.lang.Math). PI }" />
```

```
</bean>
```

## Diff. bet' BeanFactory & Application Context Container

BeanFactory

Never used in real project

Beanfactory

VIMP

- ① BeanFactory is basic container & it cannot Pre instantiatied Singleton beans.
- ② Beanfactory container only supports Singleton & Prototype supports.
- ③ SpEL is not supported by BeanFactory.

Thip

- ④ In Beanfactory we need to explicitly add an object of a BeanPostProcessor class to the BeanfactoryContainer.

VIMP

- ⑤ a Beanfactory container can create a Bean objects and injects dependency. but it cannot provide other additional services like transaction, scheduling, or messaging etc.

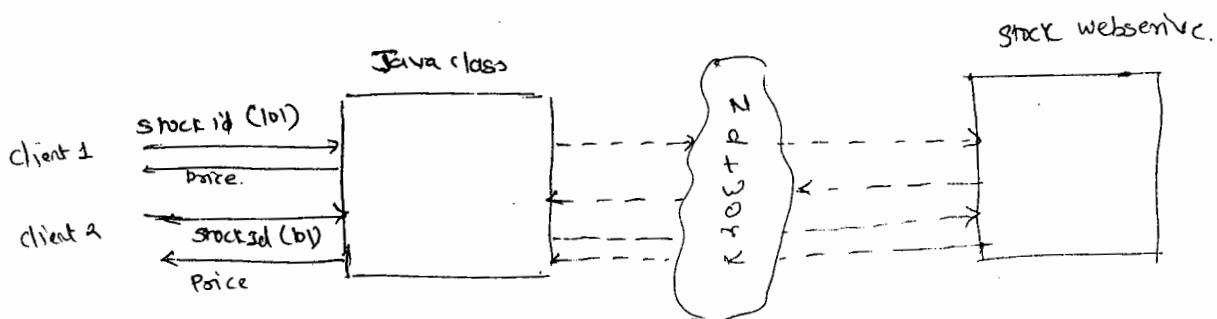
always used.

Application Context

- ① Application Context is an advanced container and it can Preinstantiate Singleton bean.
- ② ApplicationContext Container supports all beans scope.
- ③ SpEL is supported by Application Context.
- ④ In ApplicationContext, we need to configure the BeanPostProcessor class in XML. but we no need to add an object explicitly.
- ⑤ ApplicationContext container will inject dependencies and also provides the additional services to the beans.

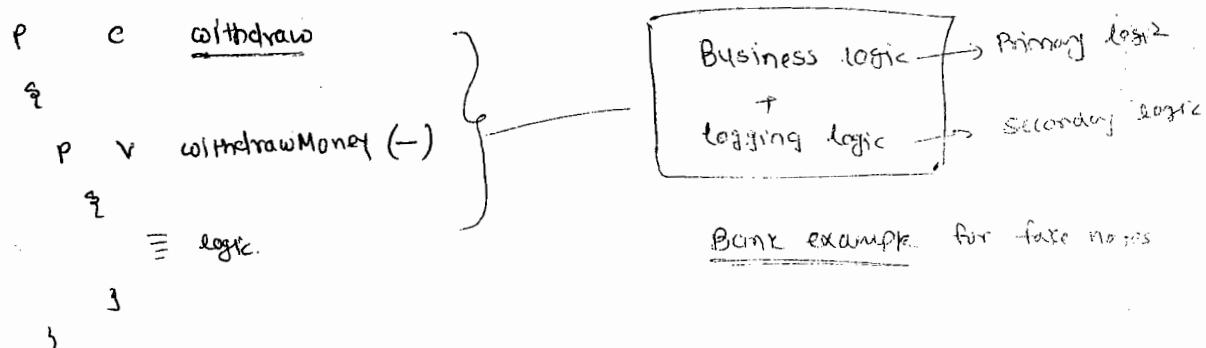
# Aspect Oriented Programming (AOP)

Primary logic & secondary logic



Primary logic → business logic @

Secondary logic → caching logic ↑ to improve performance



→ Aspect oriented Programming

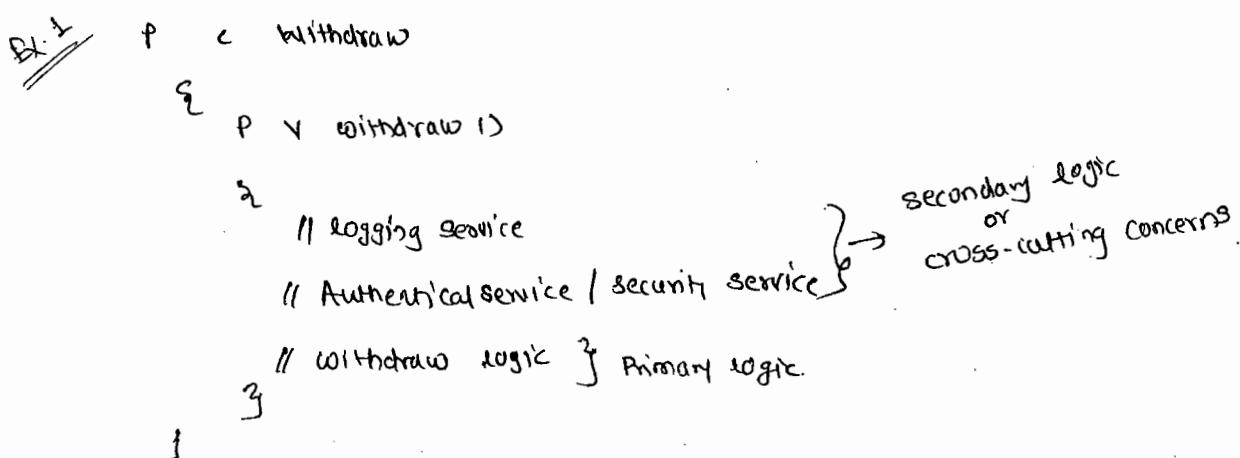
→ when creating Java Appl' with busi logic, it contol consist both Primary and Secondary logics. a Primary logic is a business logic and secondary logic is a service logic to a business logic, to make it as efficient.

for example

- ① → we have created a Java class, it reads a stockId from client then calls a web service in network to read the price and then returns to the client. this is business logic in the class.
  - ② → if other clients are also sending same stockId then reading the price everytime by calling the webservice will make the performance of an application is poor.
  - ③ → To improve the performance we can add some caching logic to the business logic. Here business logic is primary & secondary logic is caching

→ a primary logic can function on its own but a secondary logic will not function on its own

→ In Spring technology terminology the secondary logic is called cross-cutting concerns.



Ex.2

P C Deposit

§ P ✘ deposit()

```

 {
 // Logging Service
 // Authentication Service
 // deposit logic
 }
}

```

} cross-cutting concerns

- Mostly same ~~class~~ cross-cutting ~~as~~ logic are required multiple business logic in an application.
- if Cross-cutting logics are duplicated then we will get redundancy & also code maintenance problem.
- In order to overcome above problem we separate the cross-cutting logics from business logic of Application.

Q. What is backend OS? → file system.

Q. diff b/w AOP & OOP → AOP is way of Achieving OOP  
Internally AOP is also uses OOP

IS-A → tight coupling  
HAS-A → loose coupling

16-3-2015

- By using Object oriented Programming (OOP) relationships like IS-A & HAS-A, we can combine services to the business.
- In IS-A relationship, a business class extends services class.  
we find two problems here.

- ① If a business class extends services class then it can't extends another business class.
- ② If we forgot to call a service then a business will loose that service.

~~16-3-2015~~

- In case of a Has-A relationship, we need to create an object of services class in each business class. We have a problem in this HAS-A relation
  - (i) if we forgot to call a service on a business then it will loose that service
- To resolve the problem of manually applying IS-A & HAS-A relation between business & service classes, spring framework has introduced another style of programming called AOP.
- In case of AOP, the spring container will generate a proxy class per business class for combining the services to the business. For this we need to tell the container about what services are needed to what business by defining through an XML Configuration file.

#### → AOP Terminology:

\* **Aspect**: It is a term used to denote a particular cross cutting concern.

For example: We call login service as logging aspect

#### **Advice :-**

- An advice is an implementation of aspect.
- An advice will add some new behaviour to the business.
- The implementation code of a logging aspect is logging

#### **Join Point**: → nothing but the method

- It is a point at which services are going to be combined with a business.

→ In spring the services are combined with a business when that business method is accessed. So it is called a Method access join point.

### PointCut

- a PointCut is an expression, which is used to put a condition for identifying JointPoints.
- So finally, a PointCut is a group of Joint Points.

### Advisor

- A PointCut expression only identifies JointPoints, but a PointCut doesn't know what advises are required for the jointPoints.
- An advisor combines an advice to a PointCut. It is also called a PointCut advisor.

$$\text{adviser} = \text{PointCut} + \text{advice}$$

### Weaving

- It is a process of converting a target class into Process class by adding advised services.
- Before adding services through AOP, a business class is called a target class & after adding the services through AOP a business class becomes a proxy class.
- PreAOP a business class is target
- PostAOP a business class is proxy
- This weaving can be done at runtime by a weaver class.

In Spring framework AOP application can be developed in 3 approaches

- ① Proxy-style AOP [1.x]
- ② XML-based AOP [2.0]
- ③ Annotation-based AOP [2.5]

### ① Proxy-style AOP

#### Creating advices

- this style of AOP is introduced with Spring 1.x. the
- the implementation of a service will be defined in a advice.
- Services are need to be injected to the business logic at 4 points
  - ① before a business logic starts. it may need a service.
  - ② after a business logic is executed, it may need a service.
  - ③ a service may be needed when an exception is occurred in the business logic.
  - ④ a service may be needed before business logic starts. & after a business logic is executed.
- Based on the places in a business for injecting a service, advices are divided into 4 types
  - ① before advice.
  - ② after returning advice.
  - ③ throws advice.
  - ④ around advice.

## ① before advice

- a Service logic which we want to apply ^, before business logic going to start is defined in before advice by implementing our class from method before MethodBeforeAdvice interface.
- MethodBeforeAdvice has one abstract method called "before". so we should override this method to implement a service
- During compilation time the services will not be applied to our logic, services will apply only at run time.
  - so we need to implement our class to MethodBeforeAdvice interface
- for ex:
- ```
public class LoggingBefore implements MethodBeforeAdvice {
```

{

@Override

```
public void before(Method m, Object[] args, Object target)
```

{

=

}

y

- by using Method class object, a service can access a name of business method.
- by using Object[], a service can access a business method Parameter.
- by using target object, a service can access helper methods of Business Class.

②

After Returning Advice

- the services which we want to apply or to a method, after a business logic is executed, will be defined as an after returning advice.
- an after returning advice can be created by implementing our class from AfterReturningAdvice interface.

Public class LoggingAfter implements AfterReturningAdvice

{

③ Override

Public void afterReturning (Object returnValue, Method m,
Object[] args, Object target)

{

=

g

}

- a service defined in afterreturning advice can access the returnvalue of the business method.
- If a business method has return type as void then the returnvalue in the advice will be null.

③

throws advice

- if ~~we~~ want to apply a service to a business logic, when an exception is occurred in business, then we need to ~~the~~ create ~~a~~ throws advice.
- To create a throws advice, we need to implement our class from `ThrowsAdvice`.
- `ThrowsAdvice` is a marker, so it doesn't have any abstract methods.
- we must defined the implementation of service in `afterThrowing()` method only because a weaver class in Spring defined in such a way that it calls after throwing method of a throws advice.

for Ex

P C Loggingthrows implements ThrowsAdvice

Marker interface

P v `afterThrowing (Method m, Object[] args, Object target,`

`Exception e)`

1

2

3

4

④ Around advice

before advice + after returning advice

- an around advice is a combination of before advice & after returning advice
- this around advice type is included by spring framework from another open source group ~~called~~ AOP-alliance group.
- to create an around advice, we implement our class from an interface MethodInterceptor
- MethodInterceptor interface has one abstract method called around(), so we need to override around() method, to define before & after returning services or advices.

Public class LoggingAround implements MethodInterceptor

{

@Override

Public Object around (MethodInvocation mi)

{

= } before

Object o = mi.proceed();

= } after-returning

return o;

{

}

- In the middle Proceed method is called to invoke the business method.
- the return value of Business method will be stored in Object. if a business method has return type as a void then null will be stored in an object

Defining Pointcuts:

- In spring framework, jointpoints are methods
- all business methods of class may not require advices so a pointcut identifies a collection of jointpoints based on some condition
- Spring framework has already given some predefined Pointcut classes, to configure the conditions. if they are not suitable to our project then we can define custom pointcut class.

```
String mappedName;
String[] mappedNames;
    \ configure with
    <list> tag
```

- the 2 Predefined Pointcut classes given by fw

- ① NameMatchMethodPointcut
- ② JdkRegexpMethodPointcut

- with NameMatchMethodPointcut class, we need to configure either MappedName or MappedNames property.
- To configure a single condition for identifying jointpoints, we configure MappedName property
- for multiple conditions, we configure MappedNames property

Example 1

```

<bean id = "id1" class = "NMMP">
    <property name = "MappedName"
        value = "Say * />
</bean>

```

void sayHi() {

=

g

void welcome()

=

g

void sayBye()

=

g

Example 2

```

<bean id = "id1" class = "NMMP">
    <property name = "MappedNames">
        <list>
            <Value> say* <Value>
            <Value> *e <Value>
        </list>
    </property>
</bean>

```

→ In Example 1 a Pointcut makes methods starts with say as joint points. in example 2 Pointcut makes methods starts with say and ends with e as joint points.

→ In Name match "NameMatchMethodForIt" we can use only one special character "*" and it can be used at end of values or at begin of value but not in middle

→ MappedNames is a property of type String[] so it is configured in XML with list tag

*→ with JdkReg [Jdk Regular expression Method Pointcut] we can configure Pointcut conditions effectively, with the help of regular expression symbols

→ each Regular expression value must begin with ".*"

→ we can use the following special character to define condition.

- * → replaces 0 or more characters.
 - + → replaces 1 or more characters
 - ? → replaces 0 or 1 characters.
- [] → for range of characters

Example 1

```
<bean id = "id1" class = "JRMP">
  <Property name = "pattern">
    value = "* say. + [0-9] |
      ↳ starts with say & ends with
        (0-9) number.
```

Example 2

```
<bean id = "id2" class = "JRMP">
  <Property name = "patterns">
    <list>
      <value> * say. + [0-9] </value>
      <value> * [a-z] * [a-e] </value>
    </list>
  </Property>
</bean>
```

- StaticMethod MatcherPointcut }
 → DynamicMethod MatcherPointcut }

Creating a Custom Pointcut

- if predefined Pointcut classes are not enough then we can define user defined pointcut classes. they are called Custom Pointcut.
- In Proxy style AOP Pointcut classes are two types
 - (1) Static Pointcut.
 - (2) Dynamic Pointcut.
- a Static Pointcut identifies a jointPoint, based on class name & method name but not on Parameters values of the method.
- a dynamic Pointcut identifies jointPoint based on class name, method name and also Parameters values.
- if we want to create user define static Pointcut class then we need to extend our class from StaticMethodMatcherPointcut

public class MyPoint extends StaticMethodMatcherPointcut
 (Abstract)

④ Overide

public boolean matches (Method m, Class c)

?

≡

g

3

- If we want to create a user define dynamic Pointcut class then we need to extend our class from DynamicMethodMatcherPointcut class

Public class MyPointcut extends DynamicMethodMatcherPointcut {
(Abstract)

② override

p boolean matches (Method m, Class c, Object[] args)

{

=

g g

Configuring advisor

→ an advisor will attach advices to a set of jointpoint identified by Pointcut.

→ an advisor is a combination of Pointcut & advice.

→ framework has given predefined advisor class called DefaultPointcutAdvisor and we need to configure this class in Spring configuration file

```
<bean id = "advisor" class = "DefaultPointcutAdvisor">
```

```
  <Property name = "pointcut"  
            ref = "pt1"/>
```

```
  <Property name = "advice"  
            ref = "adv1"/>
```

ProxyfactoryBean → act as Weaver class

→ Create proxy class at run time & create proxy class object

Weaving → convert target class to proxy class

→ Create proxy class at runtime & add business & service logic at runtime
↳ by weaver class

ProxyfactoryBean → is Weaver class.

① ProxyInterfaces

Configuring Weaver

- a Weaver class creates a Proxy class at run time for combining the services with a Business logic.
- when creating a Proxy class at run time, that Proxy class is implemented from an interface i.e implemented by target class (Business class).
- In Spring a bean which returns an object of another bean is called a factoryBean.
- a ProxyfactoryBean is Weaver class given by Spring framework, which creates a Proxy class & returns an object of the Proxy class.
- with Proxy fact ProxyfactoryBean class, we need to configure following 3 properties.

① ProxyInterfaces → Interface name.

② InterceptorNames → id's of advisor

③ target → id of target class.

```
<bean id = "pfb" class = "ProxyfactoryBean">  
  <p name = "proxyInterfaces" value = "Demo"/>  
  <p name = "interceptorNames">  
    <list>  
      <value> advisor1 </value>  
      <value> advisor2 </value>  
    </list>
```

<1P>

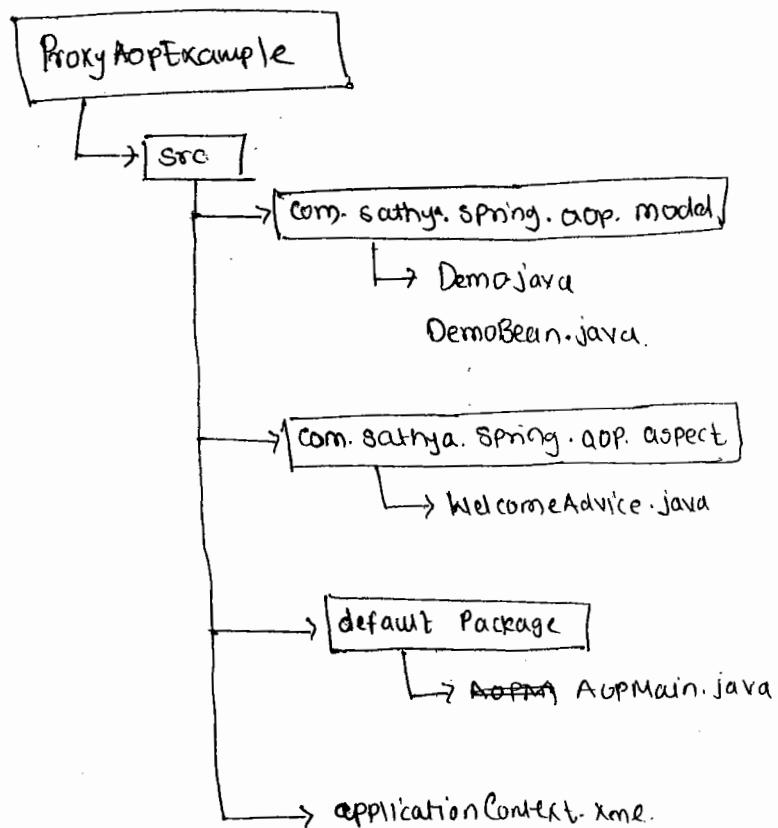
<p name = "target" ref = "db" />

Q <bean>

20 - 3 - 2015

Q. what is factory Bean ?

→ Produces object of another Bean class

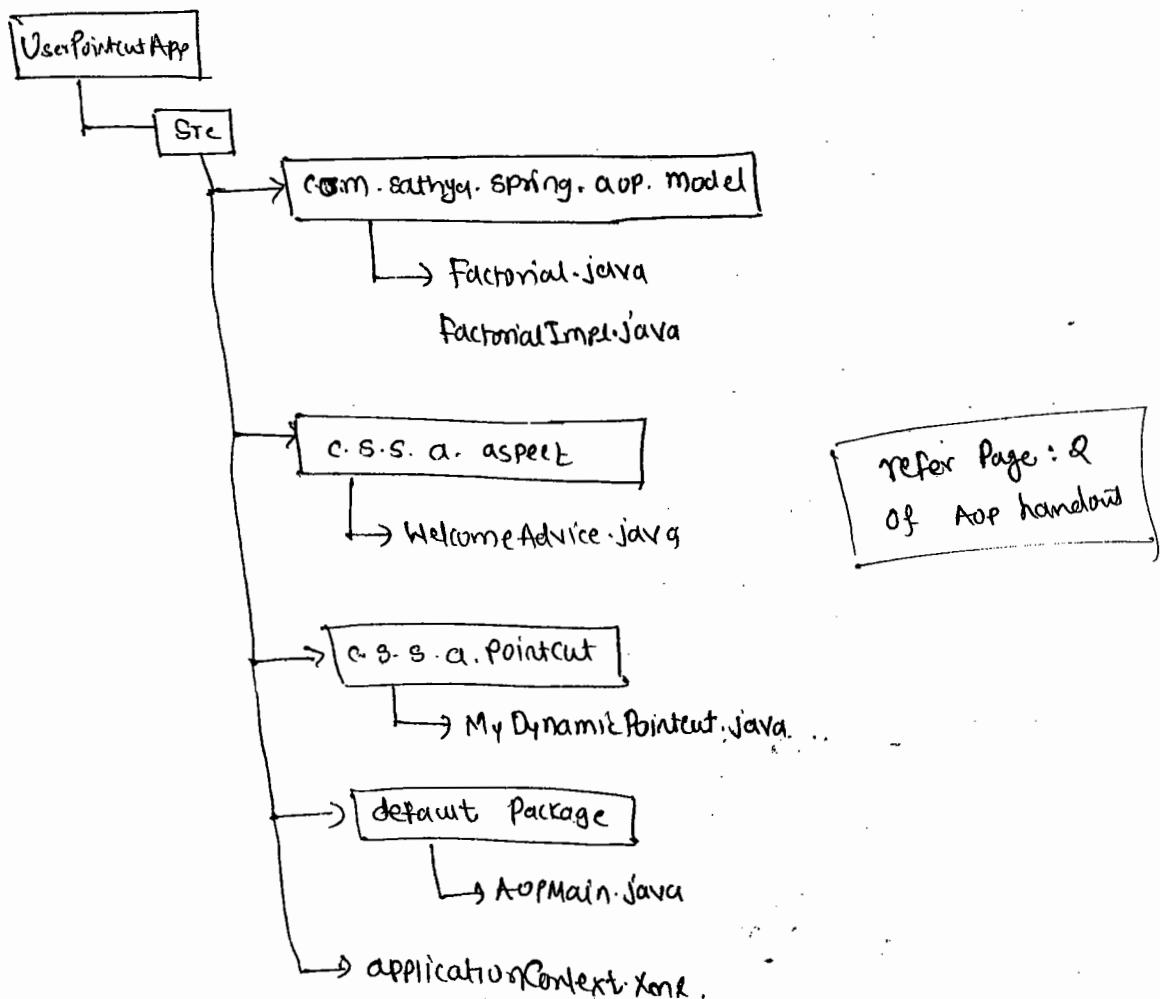


JAR files to build Path

- commons-logging
- spring-expression-4.0.5.RELEASE
- spring-aop-4.0.5.RELEASE
- spring-beans
- spring-context
- spring-context-support
- spring-core
- aopalliance-1.0

Example on User defined Pointcut

→ In the below example, we are creating and user defined Pointcut it makes a method as Pointcut if its parameter value greater than 6



- (1) Proxy-style [1-x]
- (2) XML-based [2-o]
- (3) annotation-based [2-s]

② * XML-Based AOP *

→ the following ^{are the} problems identified in Proxy-style AOP

- ① In proxy-style, to create an advice, our class must implement a predefined framework given interface. so application is tightly coupled with framework.
 - ② for creating multiple services, we need to create multiple classes. so the number of classes are increased.
 - ③ All advice classes are configured in XML file. so the number of objects are increased.
- To overcome the issues raised with the Proxy style AOP, we got XML-Based AOP in Spring 2.0.
- In XML-Based AOP we can define all advices ~~class~~ in one class, so the number of classes are reduced.
- a class used to define advices is a POJO class, so it is not tightly coupled with framework
- Because of one advice class, the no of objects are decreased.

Creating Advices in XML-Based AOP

- In XML based AOP a new type of advice is added called after or finally advice
- after returning advice will be executed if business method is executed or completed without exceptions, similarly a throws advice is executed, when a business method raises an exception.
- after advice is executed when a business method is completed without exception or with exception it means this after advice is like a finally block
- In XML-based AOP we have totally 5 types of advices.

- ① before advice.
- ② after-returning advice.
- ③ after-throwing / throws advice
- ④ after / finally advice.
- ⑤ around advice.



JoinPoint → Interface
Extends
ProceedingJoinPoint → Interface

- ① → When defining advices, if Business method name and Business method arguments are needed to define the service logic, then they are accessible through JoinPoint parameter.
- JoinPoint is an interface of AspectJ language
- In XML Based & Annotation Based AOP our spring framework takes the support of AspectJ language.

```
public class MyAspect
{
    public void loggingBefore(JoinPoint jp)
    {
        Method m = jp.getSignature(); → return Method object
        String str = m.getName();
        Object[] args = jp.getArgs();
        ...
    }
}
```

28-3-2015

- ② → When we are creating afterReturning advice, if we want to access a business method return value then we need to take the a parameter of type "Object" in the advice method.

```
public class MyAspect
{
    public void loggingAfterReturning(JoinPoint jp, Object o)
    {
        ...
    }
}
```

I
It stores return
value of a
business method.

③ → when we are creating "after-throws" advice, if we want access the Exception details occurred / raised in business method then we need to take Exception or Throwables as parameter to the throwsAdvice() method.

```
public class MyAspect
```

```
{
```

```
    public void loggingAfterThrows (JoinPoint jp, Exception e)
```

```
{
```

```
    ==
```

```
3
```

```
}
```

④ If we want to define a "finally-advice" then it can only access the Method name and arguments of the business method, but it cannot access return value of a business method.

```
public class MyAspect
```

```
{
```

```
    public void loggingAfter (JoinPoint jp)
```

```
3
```

```
==
```

```
4
```

```
4
```

⑤

→ when defining a "around" advice, the method must take
ProceedingJoinPoint object as a parameter, because
In the middle we need to call the Proceed() method in order to
call the business method.

public class MyAspect

{
 Public Object around (ProceedingJoinPoint pjp)

}

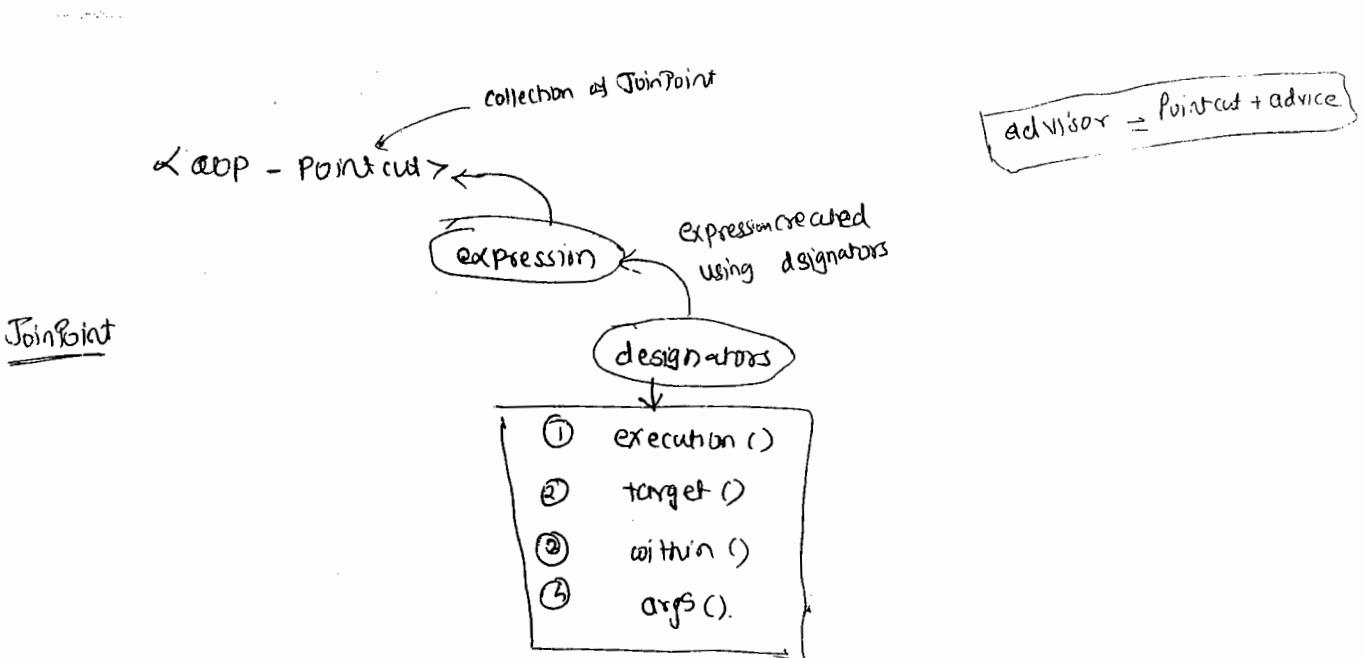
=

Object o = pjp.Proceed();

=

{

g



Configuring Pointcuts :

- Pointcuts identifies JoinPoints based on condition
- In an application there are multiple classes and each class can have multiple methods. but all classes and the all methods does not required advices. so a Pointcut identifies JoinPoints methods based on condition.
- in XMLE based AOP Pointcuts are Expressions, they are constructed using designators and configure using `<aop:pointcut>` tag.

`<aop:pointcut id = "pt1"`

expression = "Pointcut expression" />

- Pointcut expressions are constructed using the following 4 designators

① execution	4 designators
② target	
③ within	
④ args.	

- Pointcut identifies JoinPoint
- Joinpoint is method
- methods declared in expression

① Execution () designator:

- ~~these~~ this designator is used for passing a condition to identify the JoinPoints, as an expression.
- a condition expression follows the below syntax.

[modifier] returntype method-name-pattern (params - pattern)

optional

Example 1

< aop: pointcut id = "pt1"

expression = "execution (public double com.sathyaspring.Account.*(..))"

Account → may be Interface or class.

→ According to the above expression methods of account with return type double, with public modifier and having any parameters are identified as JoinPoints.

Example 2

< aop: pointcut id = "pt2"

expression = "execution (* com.sathyaspring.*.*(..))" />

any no. of parameters

Package Class Methods

→ ~~below~~ according to the above condition or expression, all the methods of all the classes of com.sathyaspring package are identified as JoinPoint

Example 3

< aop: pointcut id = "pt3" />

expression = "execution (int com.sathyaspring..*.* (int,..))" />

return type

→ according to the above condition the methods of classes of com.sathyaspring and its subpackages, return type int and first parameter is int are identified as JoinPoints.

Example 4

```
<aop: pointcut id = "pt1"  
expression = "execution (* *(..))" />
```

⇒ according to the above condition, all the methods of an application are identified as Joinpoints

24-3-2015

(2) Within designator

— this designator is to pass a package name or class name and the methods identified by execution designator will become JoinPoint, if they are called from either a given class or given package.

Ex.1

```
<aop: pointcut id = "pt1"  
expression = "execution(* pack1.A.*(..)) &&  
within(pack2.B)" />
```

→ according to the above pointcut configuration methods of Pack1.A will become joinpoints, if they are called from methods of Pack2.B

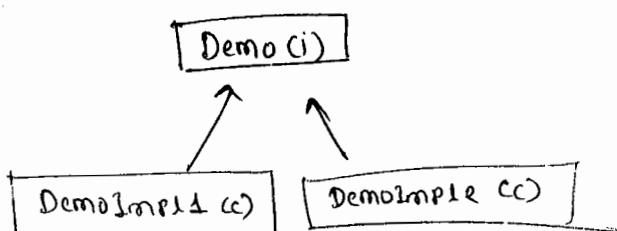
Ex.2

```
<aop: pointcut id = "pt1"
    expression = "execution (* Packt.A.*(..)) &&
        within ( com.sathya.spring.trading ) "/>
```

→ according to the Pointcut configuration methods of Packt.A will become Joinpoint, if they are called from the classes of com.sathya.spring.trading package.

③ Target

POJO / POJOD model



→ these designator is used to make methods of an interface as Joinpoint, if target object to be converted to a proxy, belongs to a given ^{target} class type.

Ex.3

```
<aop: pointcut id = "pt1"
    expression = "execution (* Packt.Demo.*(..))
        && target (Packt.DemoImpl2) "/>
```

→ according to the above pointcut configuration methods of Demo interface will become Joinpoint, if target object is an object of DemoImpl2 class.

④ args

package Pack1
Public class A

{
 p v m2 (int a, double d)

{ }
 p v m2 (int x, int y, int z)

{ }
}

every primitive, collection, & wrapper are ~~are~~ all are Serializable.

~~decor~~

→ this designator makes methods as Join Points if their arguments (Parameter) are of a given type

ex:

< aop: Pointcut id = "pt1"
expression = "execution (* Pack1.*.*(..))

&& args (java.io.Serializable)" />

→ according to the above Pointcut configuration, the methods of classes of Pack1 are ~~are~~ going to become JointPoint if ~~they~~ their argument are of type Serializable.

→ In Java by default all primitive type, wrapper type & collection types are Serializable.

→ if we want to make an user defined type as Serializable
then we need to implement our class from `java.io.Serializable`
~~interface~~.

Configuring Advisor

→ In XML based AOP, advisors are tags but not classes
→ for five types of advises, 5 advisors tags are given to
combine to ~~advise~~ advise with pointcut.

- ① <aop: before>
- ② <aop: after-returning>
- ③ <aop: after-throwing>
- ④ <aop: after>
- ⑤ <aop: around>

ex1

```
<aop: before pointcut-ref = "pt1">  
  method = "loggingBefore" />
```

→ this advisor tag combines before advice to the pointcut.

ex2 <aop: after-returning pointcut-ref = "pt1">
 method = "loggingAfterReturning"
 returning = "o" />
 ↳ parameter to store the return value
 of a business method

Ex.3

<aop: after-throwing pointcut-ref = "pt3"

method = "loggingAfterThrowing"

throwing = "e" />

↳ Parameter to store the exception
occurred in business method.

Ex.4

<aop: after pointcut-ref = "pt3"

method = "loggingAfter" />

Ex.5

<aop: around pointcut-ref = "pt3"

method = "loggingAround" />

Proxy Factory Bean.

<aop: aspectj-autoproxy /> ← xme file.

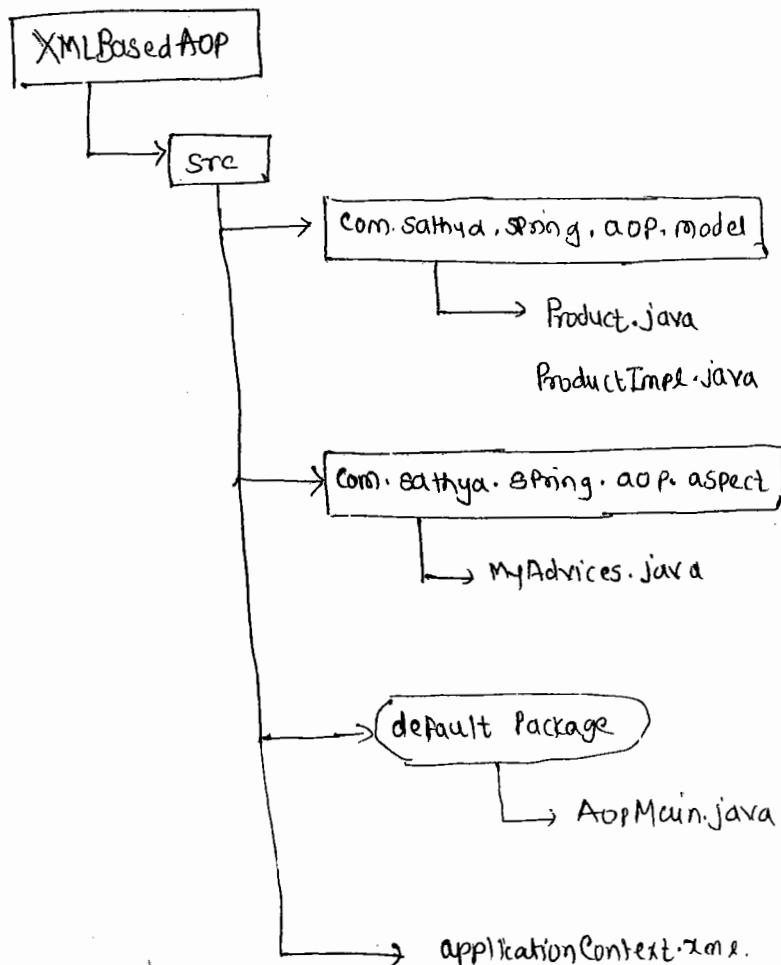
Configuring a Weaver

- In the base AOP Spring framework takes a support of aspectj aspectj. So it calls Weaver class of aspectj.
- In order to call the Weaver class of AspectJ, we need to configure the below tag in xme file

<aop: aspectj-autoproxy />

→ at run time, Spring Container calls a Weaver class
AspectJAutoProxy creates AspectJAutoProxyCreator
 to convert a target object into a proxy object.

25-3-2015



Jar file

- ① commons-logging
 - ② spring-expression
 - ③ spring-aop
 - ④ spring-beans
 - ⑤ spring-context
 - ⑥ spring-context-support
 - ⑦ spring-core
 - ⑧ aopalliance-1.6
 - ⑨ aspectjweaver-1.6.2
 - ⑩ aspectjrt-1.6.9.jar
- } newly added.

Annotation Based - AOP

- Annotations are given to cur down the AOP configuration file.
- Annotations of AOP are given by AspectJ programming language.
- at run time Spring container will convert the annotation into AOP configuration.
- there are 7 annotations to configure the AOP

① @Aspect

② @Pointcut

③ @Before

④ @AfterReturning

⑤ @AfterThrowing

⑥ @After

⑦ @Around

advisors

- to tell the Spring container that, read the information to configure AOP from a class, on top of that class we add @Aspect annotation
- @Pointcut annotation is to add a pointcut expression ~~to~~ empty method, a Pointcut can be reused by calling that empty method.
- if you do not want to reuse a pointcut expression then we no need to create empty method
- the remaining 5 ~~are~~ annotations are advisors

ex:1

② Aspect

Public class MyAspect

{

③ PointCut ("execution (* com.sathya.spring.aop.model.*.*(..))")

Public void ok() {} → (empty Pointcut method)

④ Before ("ok()")

Public void loggingBefore(JoinPoint jp)

{

--

⑤ Around ("ok()")

Public Object logging(Proceeding

public Object loggingAround(ProceedingJoinPoint p)

{

--

Object o = PJP.proceed();

--

return o;

{

}

ex.2

② Aspect

public class MyAspect

{

 ③ Before ("execution(* com.sathyq.spring.aop.model.*.*(..))")

 public void loggingBefore(JoinPoint jp)

{

}

 ④ Around ("execution(* com.sathyq.spring.aop.model.*.*(..))")

 public Object loggingAround(ProceedingJoinPoint p)

{

 Object o = pjp.proceed();

 return o;

}

diff. b/w ex.1 & ex.2

→ is in example 1 Pointcut method is created and it is called for 2 times.
but in example 2 Pointcut method is not created so expression is repeated
for 2 times.

② Before ("or ()")

↳ pointcut method.

③ After

- | | | |
|------------------|---|----------------------------|
| ② Before | → | Value |
| ② AfterReturning | → | Pointcut, returning, Value |
| ② AfterThrowing | → | Pointcut, throwing, Value |
| ② After | → | Value |
| ② Around | → | Value |

① In @AfterReturning annotations, we can pass the following 3 ^{as} elements as parameter.

- ① pointcut
- ② returning
- ③ value.

- if you want to store return value of a business method in a method parameter of an advice then we need to pass Pointcut, returning elements to the @AfterReturning annotation.
- If we do not want to store return value of business method in a advice method parameter then we directly pass value element.

Ex 1

③ Aspect

```
public class MyAspect
```

```
{ @Pointcut("execution(* com.saranya.spring.model.*.*(..))")
  public void test() { }}
```

② After returning (pointcut = "test()", returning = "o")

public void loggingAfterReturning (JoinPoint jp, Object o)

{

}

}

② → In @AfterThrowing annotation we can pass the following 3 elements as parameter.

③ 1) Pointcut 2) throwing 3) Value

→ If we want to store exception of business method in advice method parameter, then we need to pass pointcut and throwing elements. If you do not want exception of business method in a advice then we can directly pass value

Ex.1

④ Aspect

public class MyAspect

{

⑤ Pointcut ("execution(* com.*.*(..))")

public void test () { }

⑥ AfterThrowing (pointcut = "test()", throwing = "e")

public void loggingAfterThrowing (JoinPoint jp, Exception e)

{

}

}

③ → @After & @Around annotation ~~control~~ contents only a single element called value.

Q. How do you find the your java program takes too much time to execute program.

class A

{ Main

{

long x = System.currentTimeMillis(); } this called timer service.

—

—

—

long y = System.currentTimeMillis();

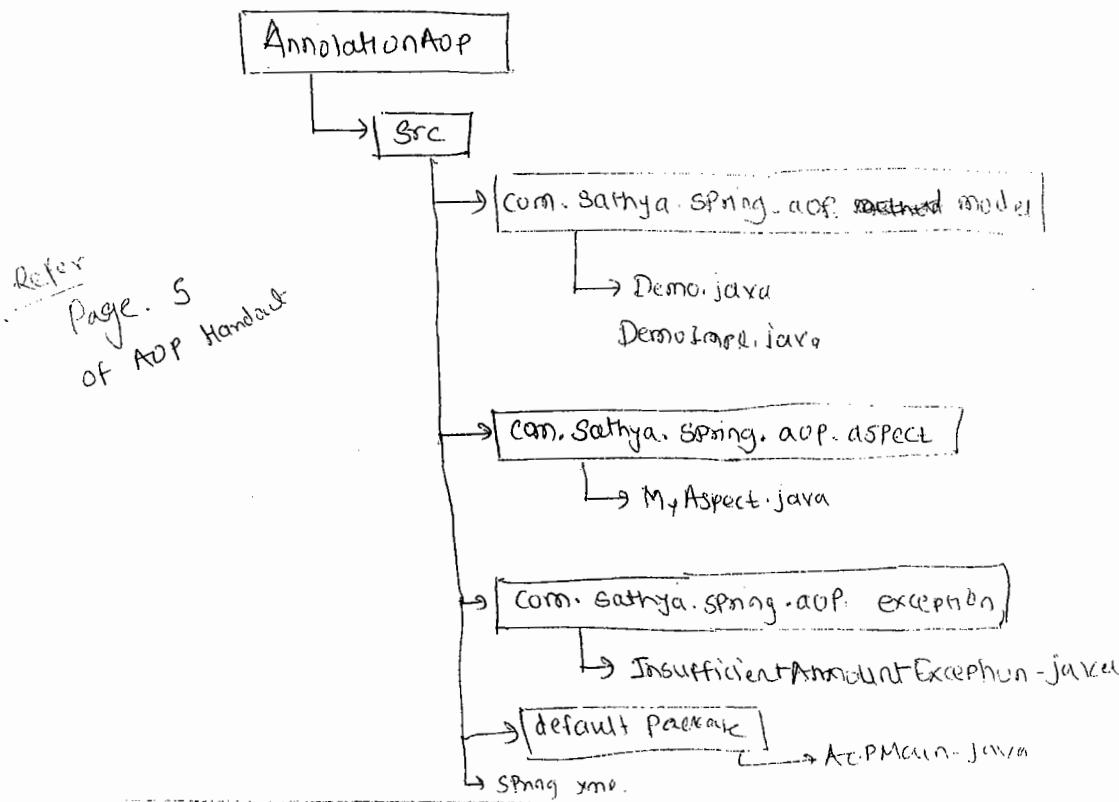
long z = y - x;

System.out.println(z);

}

}

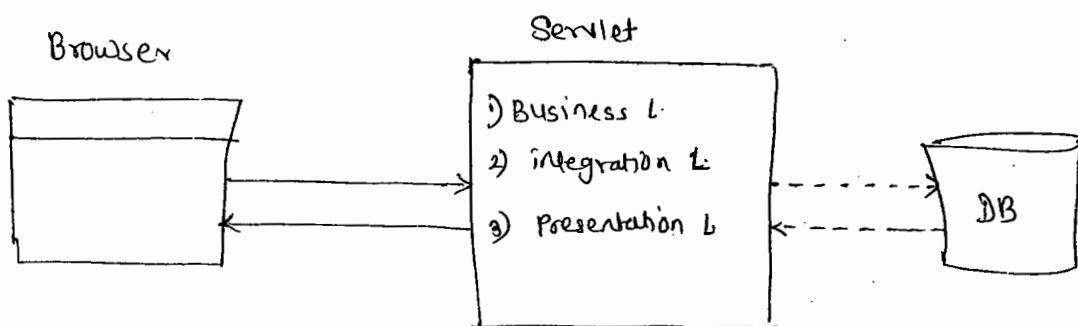
Example 4



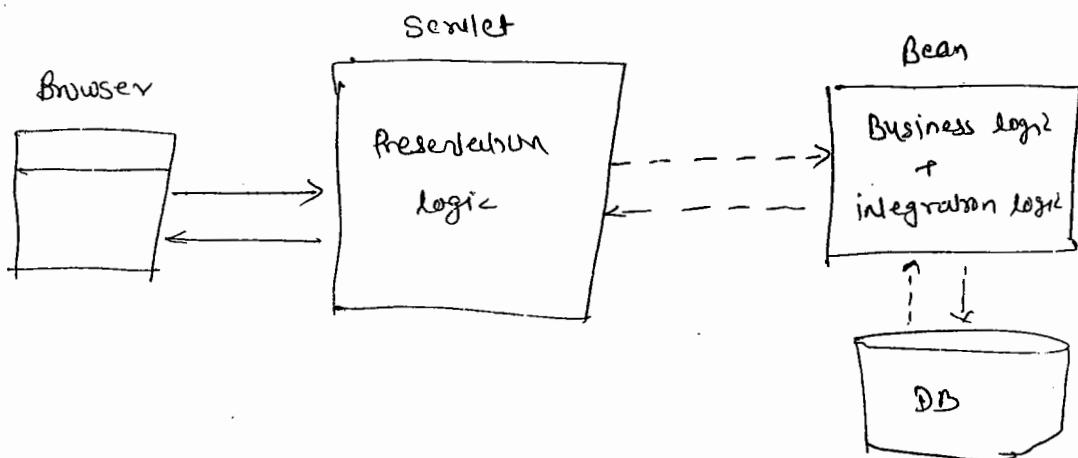
Spring MVC

28-3-2015

- In Java, web application development started with Servlet technology
In early days of web application development we used to send all
to defined business logic, integration logic & presentation logic in
a Servlet only.

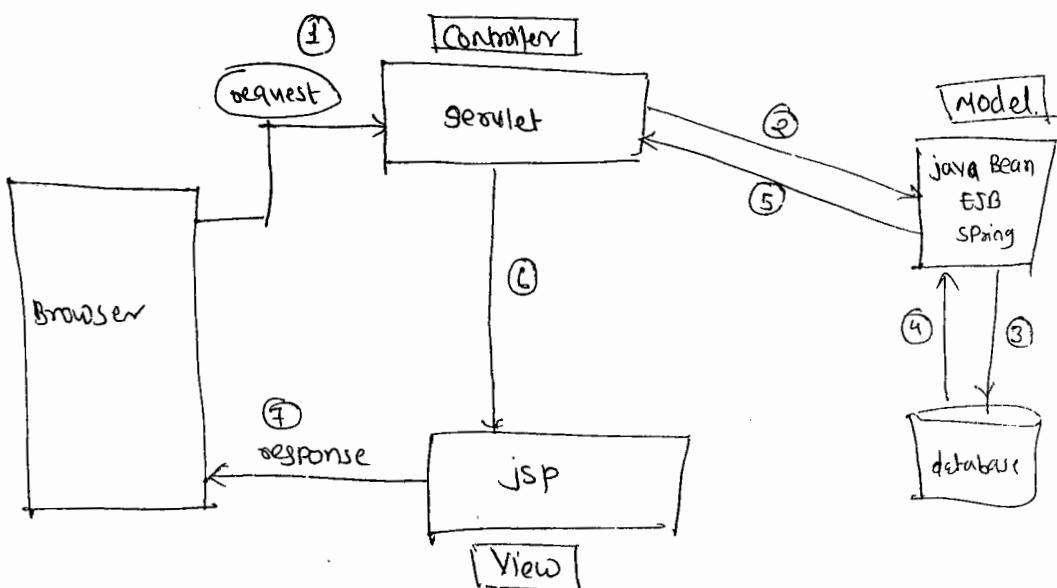


- To reduce the burden on a servlet business & integration logics
are separated from servlet to a Java bean & only presentation
logic is developed in servlet.



- The web applications which are developed with Servlet or
Servlet with Bean are called Model-1 application

- the applications developed in model-1 are going to have
 - tightly coupling between presentation logic & business logic.
- In order to provide loose coupling between business logic and presentation logic, we got Model-2.
- In Java model-2 is introduced with JSP it means model-2 applications are developed by using Servlet, Bean, & JSP component.
- a bean contains business & integration logics, a View (JSP) contains presentation logic & both are controlled by Servlet. this model 2 is also MVC Model.



- According to the MVC model, the control flow logic for all the model & view components will be defined in a **Servlet**
- defining all control flow logic in a **Servlet** has increased the burden on **Servlet**, so some helper classes are created to manage the control flow by a Servlet it is called MVC-2 model.
- In Java we have only **MVC** & **MVC-2**, but there is no **MVC-1**

→ A difference bet' MVC & MVC-2 is, in MVC control flow is define in a servlet & in MVC-2 control flow of Servlet will divided to its helper class.

Q → Why Servlet is a controller in MVC?

A → We can take a JSP as controller because JSP is aware of HTTP protocol but for control flow large amount of Java code is need to be return in a JSP. according to the Industry Standard, we can't write large amount of code in JSP so JSP is not a suitable for controller.

→ In a Bean we can write large amount of Java code but a bean is not aware of HTTP protocol, so bean is not suitable for controller, so Servlet is used as controller in MVC.

Q Why JSP is a view in MVC?

Ans We can use Servlet as view by defining presentation logic but if any changes are needed in presentation then we need to recompile a servlet reload application and restart the server. so Servlet is not suitable for view.

→ We can not define presentation logic in Java Bean because, Java Bean is unaware about HTTP protocol. so a bean is not suitable for a View.

→ So, JSP is used as view in MVC.

Q

Why Bean is a model in MVC?

- we can write business logic in a servlet but a servlet container will not ^{or provide} offer multiple services required for B-logic. So servlet is not suitable as model.
- In a JSP we cannot write large amounts of the business logic. So JSP is not suitable as model. so ~~as~~ a bean can have B-logic of MVC.

30-3-2015

Q. Explain flow of MVC! *Ans*

DispatcherServlet

Adv. Java

Servlet
Receive request
& control the flow

MVC

only receive request
Control by other helper class
or handler class

Controller responsibility



to find appropriate
handler to process
the request

Browser



Request



DispatcherServlet — receive the request & find appropriate



~~HandlerMapping~~ — handle the request
HandlerMapping based on url

→ DispatcherServlet

→ HandlerMapping

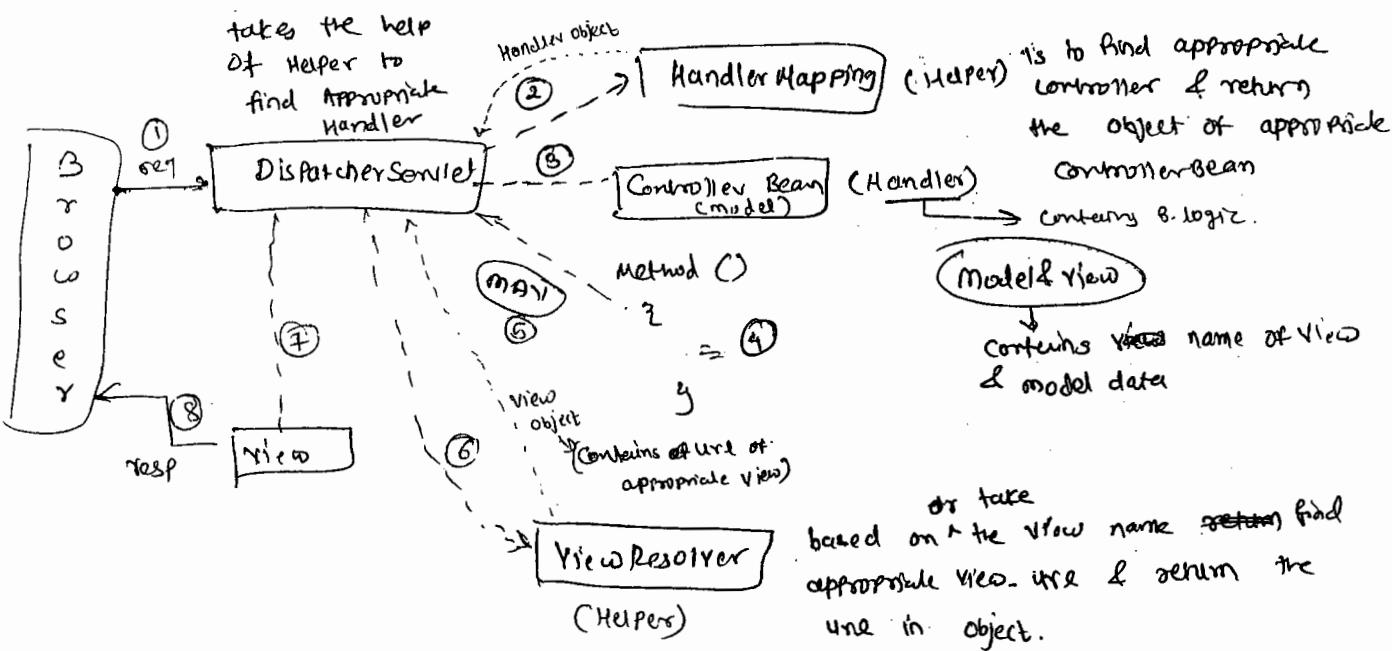
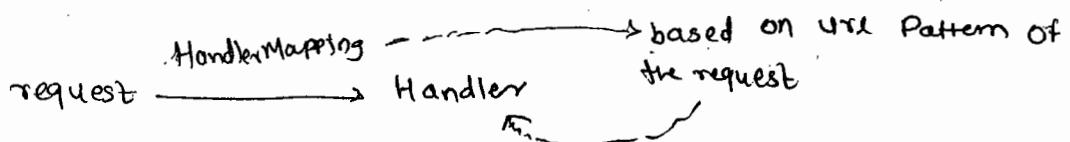
→ ControllerBean

→ ViewResolver

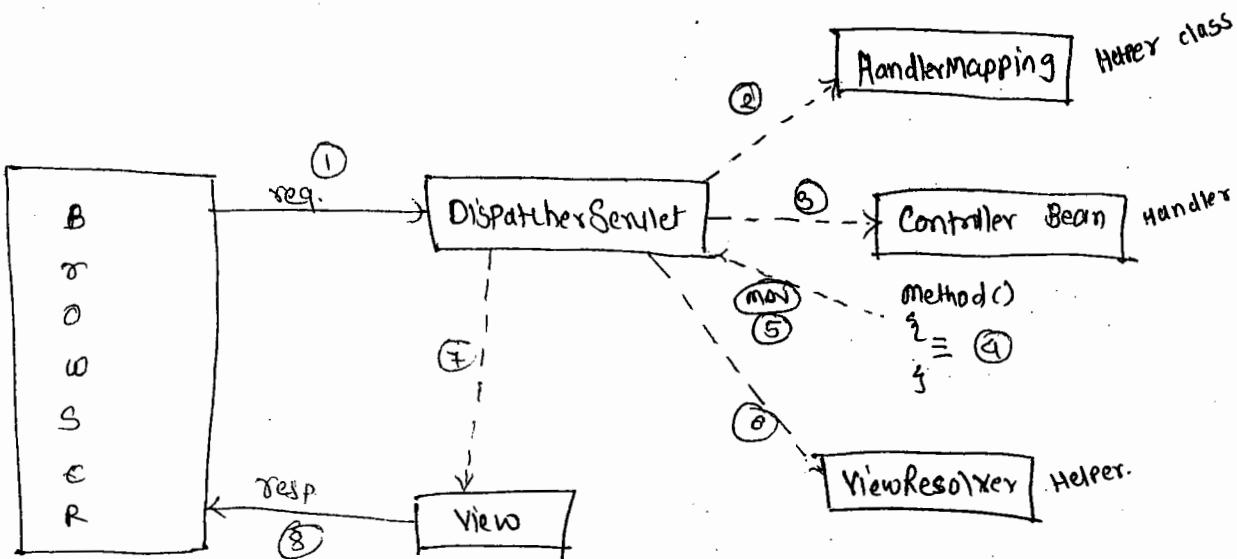
→ View

Helper classes to
DispatcherServlet

ControllerBean → Handler



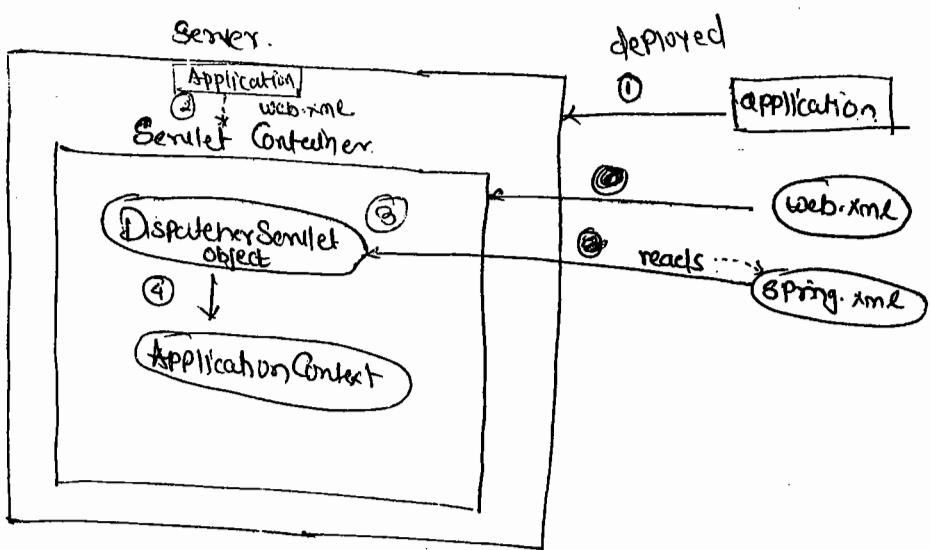
Spring - MVC flow



Step

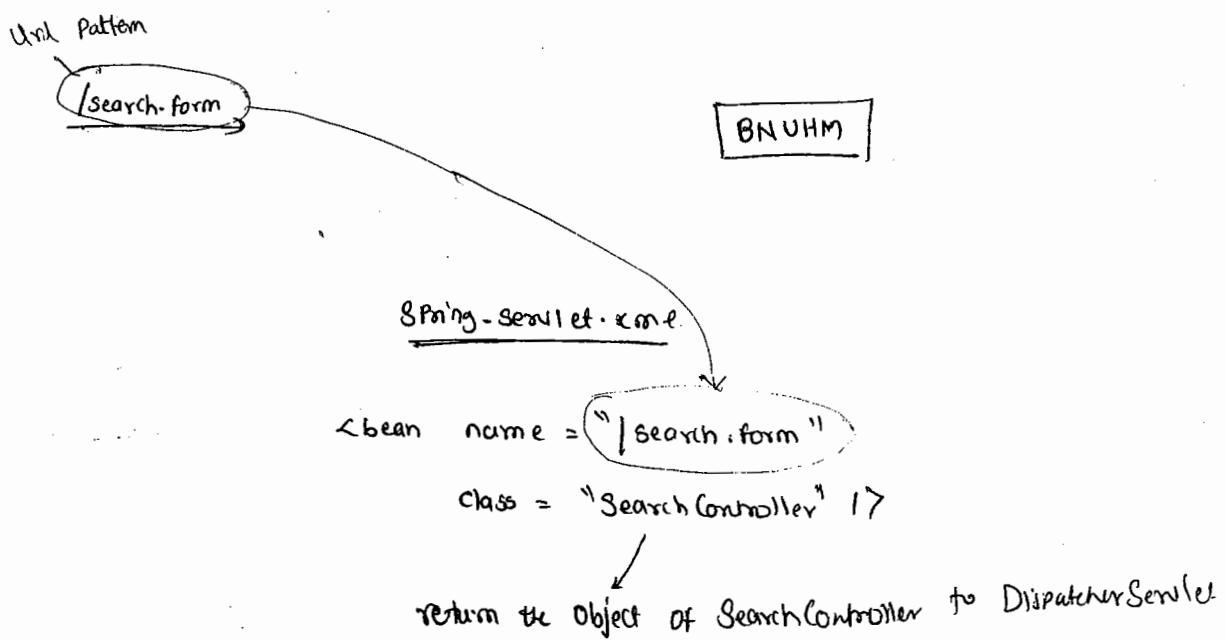
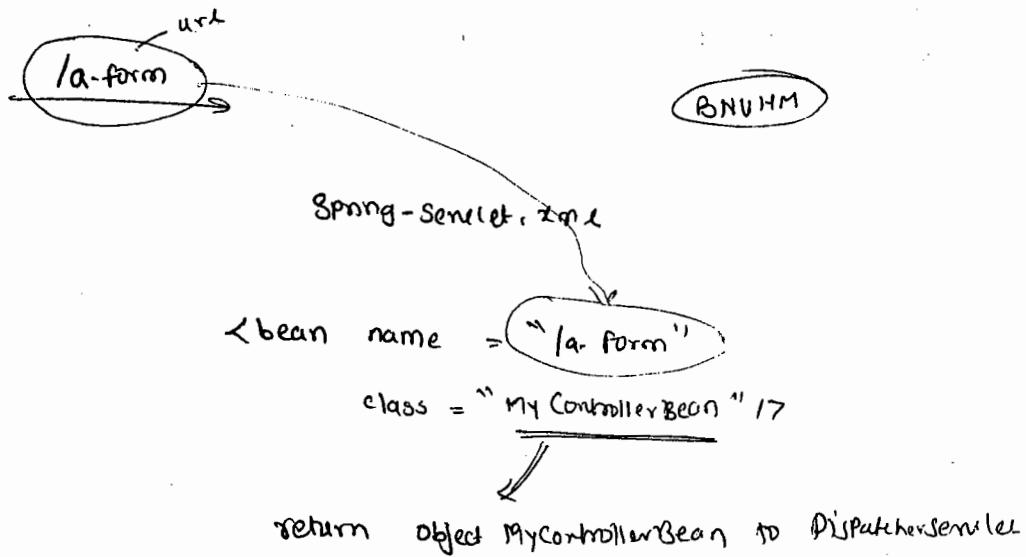
- ① a first stop for a request in spring-mvc is at the ~~Request~~ DispatcherServlet. It is the FrontController of the spring-mvc application.
- ② DispatcherServlet calls its helper class called HandlerMapping. this ~~Component~~ HandlerMapping component maps a request to a Controller Bean (request handler) based on url pattern of the request.
- ③ finally HandlerMapping returns an object of Controller Bean to the DispatcherServlet.
- ④ DispatcherServlet calls method of Controller Bean and that method will process the given request.
- ⑤ a Controller Bean returns result in the form ~~ModelView~~ ModelAndView Object. This object contains only View name or View name with Model data.
- ⑥ DispatcherServlet calls another helper class of it called ViewResolver. this ViewResolver will identify the appropriate View based on the View name and returns url of that view in the form of View object.
- ⑦ DispatcherServlet will forward a request to the view.
- ⑧ finally a view renders response to browser.

- HandlerMapping finds the appropriate controller depends on url and return the object of controllerBean to the dispatcher
- DispatcherServlet calls the methods of ControllerBean using this object and returns the view name in the form of ModelAndView object



Configuring DispatcherServlet.

- In Java MVC framework a framework only supplies a control flow logic in the form of a predefined Servlet class. e.g. in case of Spring framework it is DispatcherServlet.
- DispatcherServlet is a subclass of HttpServlet, so it can only handle HTTP Protocol request.
- we need to Configure DispatcherServlet class in a web.xml (deployment descriptor).
- when a Spring MVC application is deployed in a server, Servlet container will immediately created DispatcherServlet Object at the deployment time only. DispatcherServlet will immediately create a Spring ApplicationContext container Object, by loading Spring Configuration file.



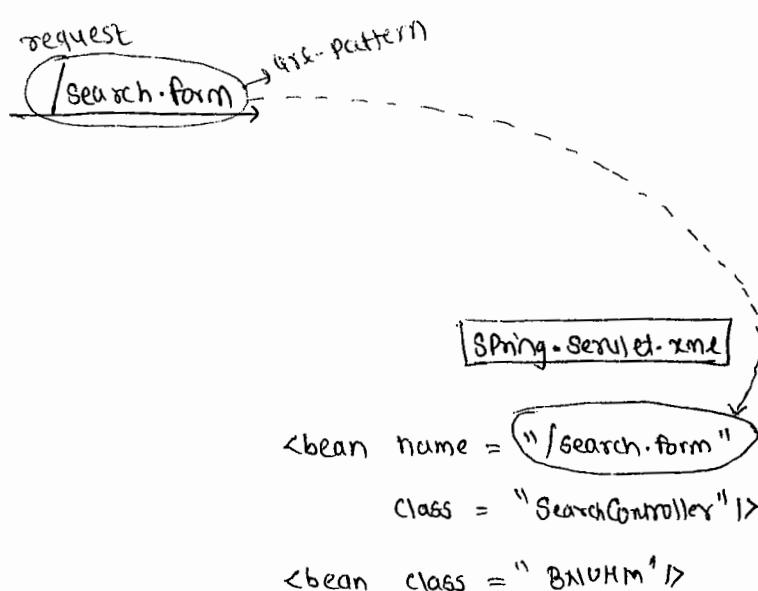
→ HandlerMapping Component

→ a Handlermapping class is Helper class to the DispatcherServlet, it will map a request to the appropriate Controller Bean based on url Pattern

→ Predefined HandlerMapping classes given by framework are

- ① BeanNameHandlerMapping
- ② SimpleUrlHandlerMapping
- ③ DefaultAnnotationHandlerMapping

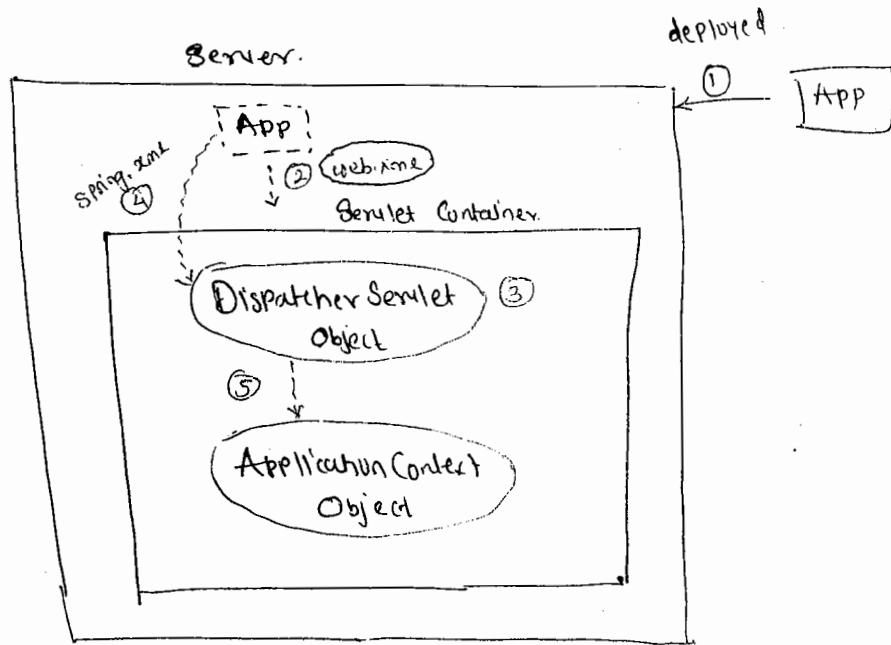
- if predefined HandlerMapping classes are not suitable for an application then a programmer can create a custom Handler mapping class by ~~ext~~ implementing HandlerMapping interface
 - BeanNameUrlHandlerMapping will ~~not~~ map a request to controller bean like the following.
 - ① BNUHM reads url pattern of the request from ~~spring~~ configuration file.
 - ② It search a bean name in spring container (spring.xml)
 - ③ which is matched with url-pattern of the request
 - ④ If matched then return a object of the controller bean class to the Dispatcher servlet.



Creating a Controller bean.

+ ① P C SearchController implements Controller:
given by framework, by using
this it tightly coupled with f/w class

✓ ② Controller
 ↑ C SearchController
 ? ↓ --- ←
 ↓ ↓ --- ←
 intermediate logic which calls
 the business logic which
 is ~~written~~



31-3-2015

one request to one servlet
for one servlet one url-pattern

exact match
directory match
✓ extension match

Types of url-Pattern
in servlet mapping

we can configure
for one servlet ~~the url-pattern~~
n number of url-pattern

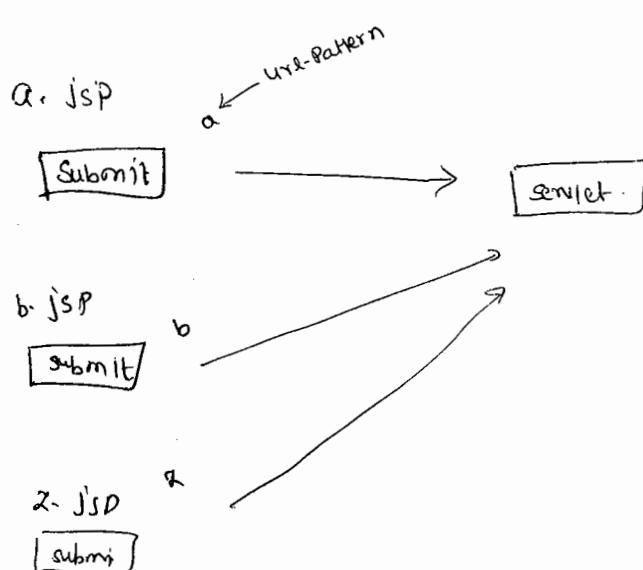
EX <S-m>
<s-n>
<u-p>/a <u-p>
<u-p>/b <u-p>

<S-m>

web.xml

/a

/b



→ DispatcherServlet should be configure in deployment descriptor like the following.

Web.xml

<web-app>

< servlet >

< servlet-name > Spring < /servlet-name >

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

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

< init-param >

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

< param-value > /WEB-INF/applicationContext.xml < /param-value >

< /init-param >

< /servlet >

< servlet-mapping >

< servlet-name > Spring < /servlet-name >

< url-pattern > *.form < /url-pattern >

< /servlet-mapping >

→ to stop each request at DispatcherServlet use
extension-match which is type of url-pattern

< /web-app >

→ if Spring configuration file name is given by following ~~some~~

< servlet-name > -Servlet.xml format then there is no need to configure
init-parameter for DispatcherServlet in web.xml.

BeanNameUrlHandlerMapping

→ reads url-pattern of request.

→ search for a

1-4-2015

{ Controller → interface
 { AbstractController → class
 { AbstractCommandController → class
 { SimpleFormController → class
 { MultiActionController → class

Stereotype Annotation

@Component
@Service
@Repository
@Controller } tells the role of the class

- A Controller bean in a Spring MVC application contains business logic for processing ~~the req~~ or intermediate logic for processing ~~the~~ request.
- One Spring-MVC App can have multiple Controller Bean classes.
- We can create a Controller Bean in 2 approaches
 - ① by implementing our class from Controller interface or by extending our class from one of the subclasses of Controller interface.
 - ② by adding "@Controller" annotation on top of the class

1st approach

public class MyController implements Controller

{
 @Override

 public ModelAndView handleRequest (request, response)

 {

 ==

 }

}

② Approach

@Controller

Public class myController

```
{  
  =  
  }  
  =
```

Model And View

Contains either view name or
view name + model data.
→ data in form of key value.

ModelAndView mav = new ModelAndView("success");

→ view name, it is logical name

ModelAndView mav = new ModelAndView("success", "msg", "Hello");
→ View name Key Value
 + {
 model data }
 Value

Key must be string
Value can be anything.

ModelAndView Object

- a Controller Bean returns ModelAndView object after processing a request
- a ModelAndView object contains either [view name] or [view name + model data]
- a Model data will be in Key/Value pair and this data will displayed on the browser by a view with some presentation.

- we can create a ModelAndView object like the following.

① ModelAndView mav = new ModelAndView("success");

→ view name also called logical name

② ModelAndView mav = new ModelAndView ("success", "msg", "Hello");

↓
View name
↓ Key
↓ Value
model data

→ We can add a multiple key|value ^{Pairs} of model data by calling addObject () method

```
ModelAndView mav = new ModelAndView ("success");
mav.addObject ("msg", "Hello");
mav.addObject ("k1", 1000);
mav.addObject ("e1", emp1);
```

Note: When adding the Model data, key must be a String and value can be any object.

View Resolver bean

→ a View resolver in Spring MVC is a helper class to Dispatcher Servlet. A view resolver finds appropriate view ^{of} rendering based on view name return by controller bean and then it returns the url of the view in the form of View object.

→ Spring framework has given 4 predefined View Resolver classes.

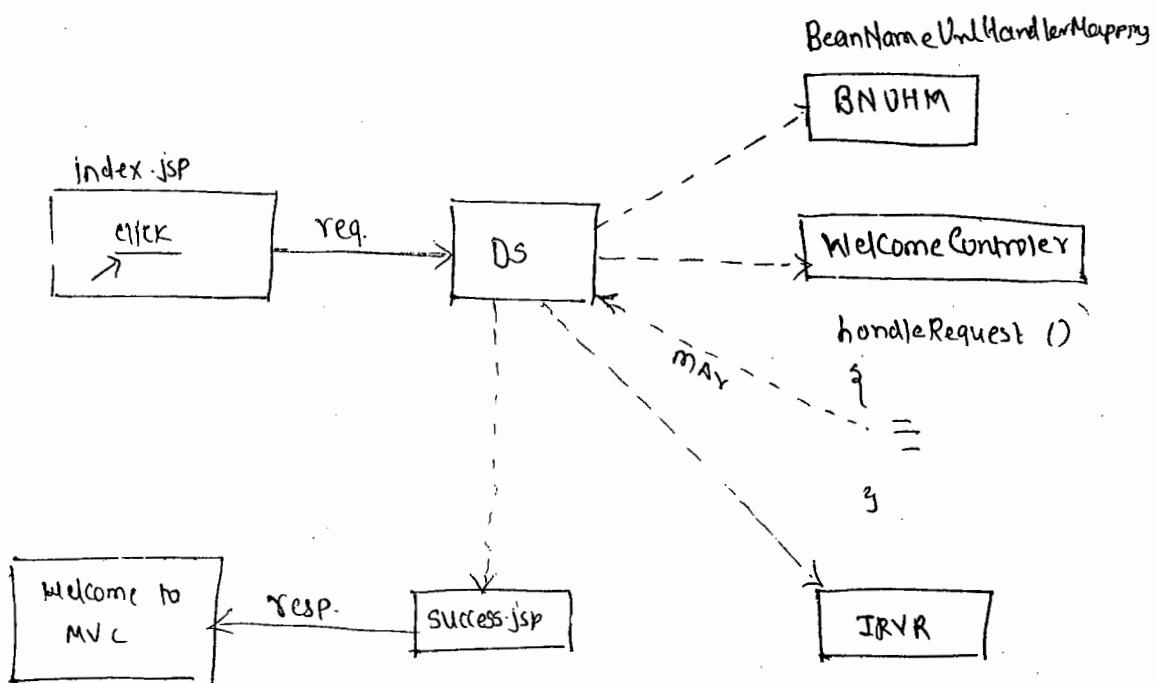
- ① InternalResourceViewResolver
- ② BeanNameViewResolver.
- ③ ResourceBundleViewResolver.
- ④ XmlViewResolver

→ if predefined view resolver classes are not suitable to a project then a programmer can create custom view resolver class by implementing ViewResolver interface.

Ex:

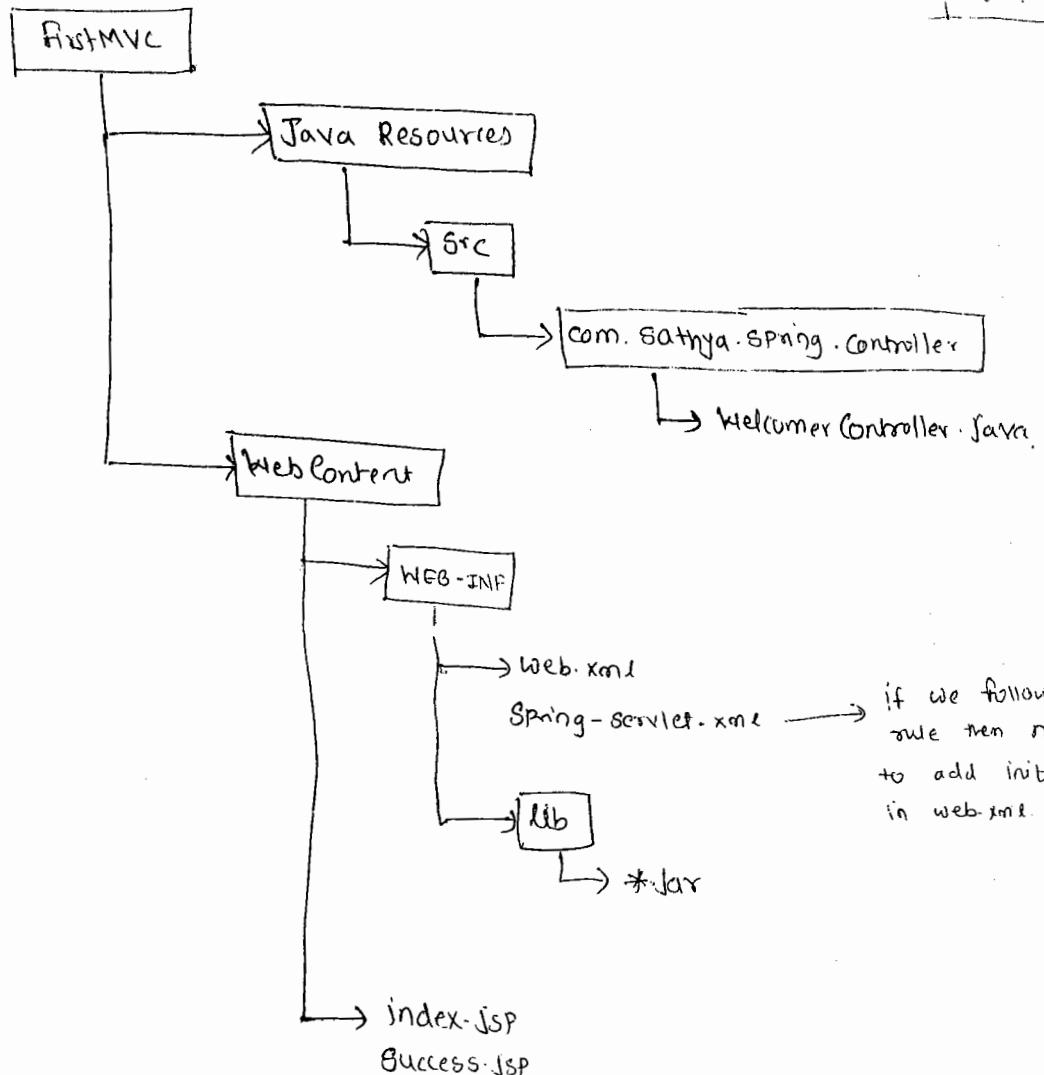
- InternalResourceViewResolver will find appropriate url of a view, by adding a prefix and suffix to the view name.
- with InternalResourceViewResolver class we need to configure prefix and suffix properties.

```
<bean id = "resolver" class = "InternalResourceViewResolver">  
    <property name = "prefix" value = "/" />  
    <property name = "suffix" value = ".jsp" />  
</bean>
```



- ① index.jsp
 - ② success.jsp
 - ③ WelcomeController.java
- ④ DS → DispatcherServlet → web.xml
- ⑤ BNUHM } → Spring Configuration file.
- ⑥ IRVR }

2-4-2015



→ Add the following list of jar to the lib

Spring-web-4.1.2.RELEASE.jar
Spring-webmvc-4.1.2.RELEASE.jar } newly added.

Commons-logging

Servlet-api

Spring-beans-

Spring-context-

Spring-context-support

Spring-core

Spring-expression.

Index.jsp

```
<center>
    <a href = "Welcome.form"> click here </a>
</center>
```

~~Web~~ web.xml

```
<web-app>
```

```
    <display-name> firstMVC </display-name>
```

```
    <welcome-file-list>
```

```
        <welcome-file> index.jsp </welcome-file>
```

```
    </welcome-file-list>
```

```
    <servlet>
```

```
        <servlet-name> Spring </servlet-name>
```

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

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

```
    </servlet>
```

```
    <servlet-mapping>
```

```
        <servlet-name> Spring </servlet-name>
```

```
        <url-pattern> *.form </url-pattern>
```

```
    </servlet-mapping>
```

```
</web-app>
```

1/ WelcomeController.java

```
Package com.sathya.spring.controller

Public class WelcomeController implements Controller
{
    return type
    Public ModelAndView method Name
        handleRequest (HttpServletRequest request, HttpServletResponse)
            throws Exception
    }

    ModelAndView mav = new ModelAndView ("success");
    return mav;
}
```

Spring-Servlet.xml

```
<beans>
```

```
<bean id = "handlermapping"
      class = "org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
```

```
<bean name = "/welcome.form"
```

```
      class = 'com.sathya.spring.controller.WelcomeController'/>
```

```
<bean id = "ViewResolver"
```

```
      class = 'org.springframework.web.servlet.view.InternalResourceViewResolver'>
```

```
          <Property name = "Prefix" value = "/">
```

```
          <Property name = "suffix" value = ".jsp"/>
```

```
</bean>
```

```
</beans>
```

Success.jsp

<%@ page %>

welcome to spring mvc!

<%@ page %>

Creating a Controller Bean with annotations.

→ the 2 min. annotations of Spring fw used is for creating a controller bean is

① @Controller

② @RequestMapping

→ If controller bean is created with annotations then DispatcherServlet calls DefaultAnnotationHandlerMapping, this handler mapping class reads url pattern of a request without extension and if it match the value with value url pattern with value of ② @RequestMapping annotation.

→ to create controller bean WelcomeController with annotations for in the above example then we need to do the following changes.

① Create WelcomeController.java like the following

Package

import org.springframework.stereotype.Controller

6

① @Controller

public class WelcomeController

{

② @RequestMapping (value = "/welcome")

public ModelAndView test()

9

return new ModelAndView ("success");

4

ModelAndView mav
= new ModelAndView
("success");
return mav;

3

- ② In `spring-servlet.xml`, configure `DefaultAnnotationHandlerMapping` & the enable Component scanning like the following.

```
<bean id = "handlermapping"
      class = "org.springframework.web.servlet.mvc.annotation.
      DefaultAnnotationHandlerMapping">

<context:component-scan base-package = "com.sathya.spring.controller"/>

<bean id = "viewresolver"
      class =
      <property>
      <property>
      </property>
</bean>
```

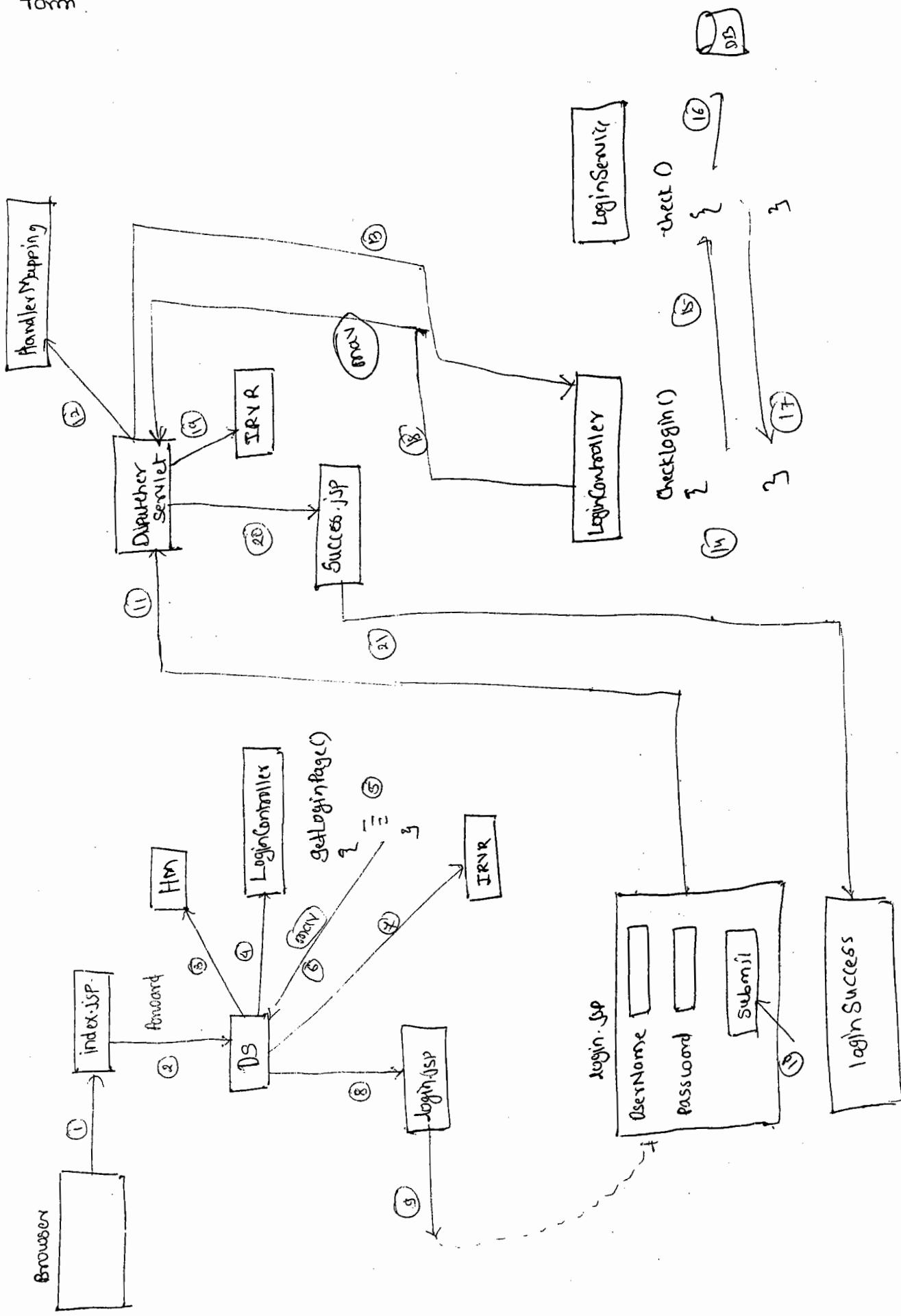
- ③ add `spring-aop` & `aspectjweaver` jar to ~~the~~ lib folder

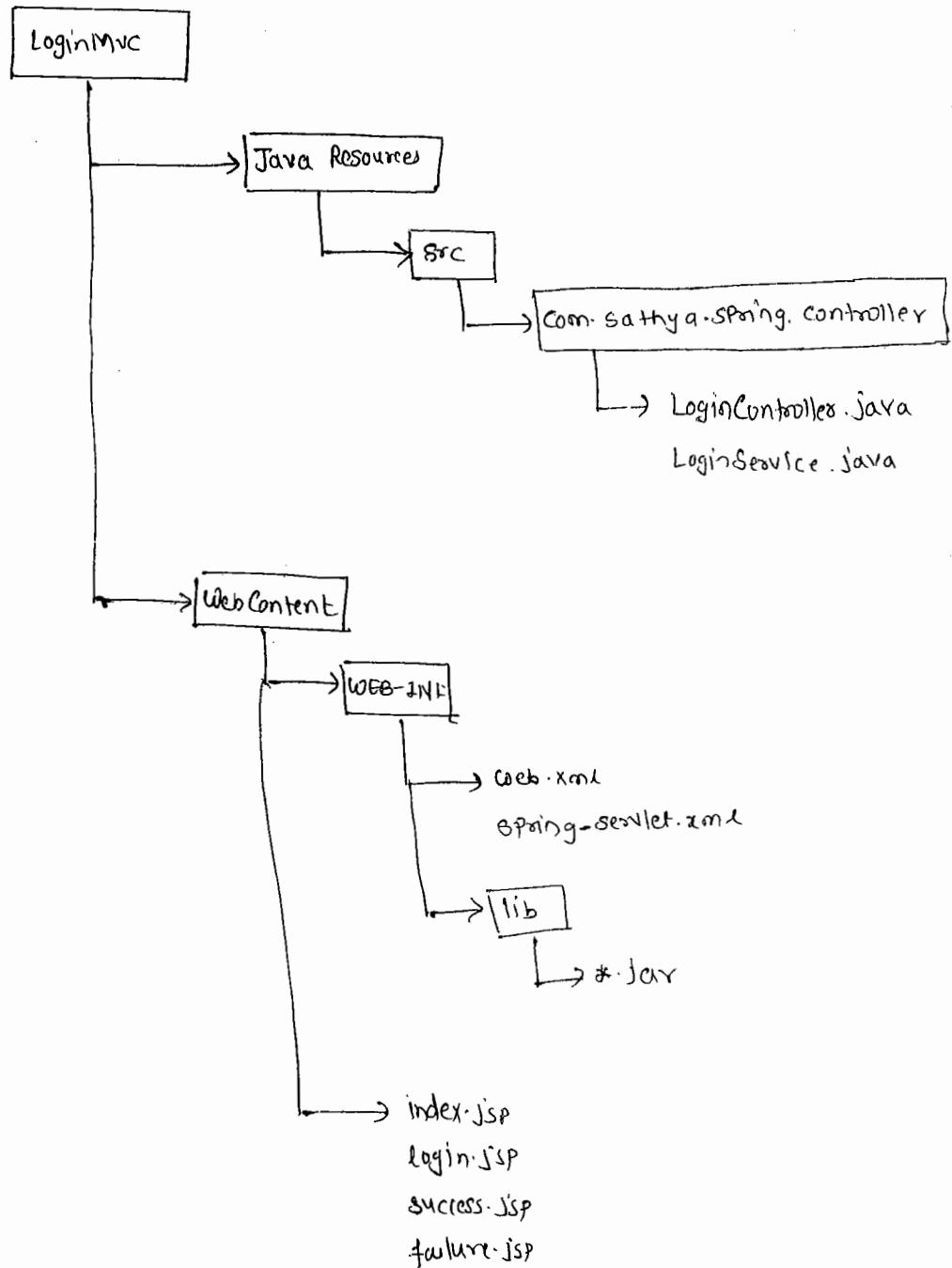
3-4-2015

Login Application with database Support

- In the following example we are creating a login page to submit a username & Password
- the controller bean of ^{this} ~~our~~ applⁿ collaborates with another bean to find whether user name & Password entered ^{by user} are valid or not.
- In spring-jdbc layer, `JdbcTemplate` class is given to perform the database operations. this template class will depend on `Datasource` object open a connection with database.
- when developing spring-mvc applⁿ it is not a recommended way to directly send a request from browser to the jsp page, which contains form design, so we need to create an index page and the through a flow of mvc we need to get the

jsp page on to the Browser, in which we design the form.





Jar files

{
 JDBC14.jar
 Spring-jdbc-4.0.5.RELEASE
 Spring-tx-4.0.5.RELEASE

 newly added.

aopalliance-3.0.0.jar
 commons-logging
 servlet-api
 spring-aop-4.0.5.Release
 spring-beans-4.0.5 RELEASE
 spring-context
 spring-context-support
 spring-core
 spring-expression
 spring-web

 spring-webmvc-4.0.5.RELEASE

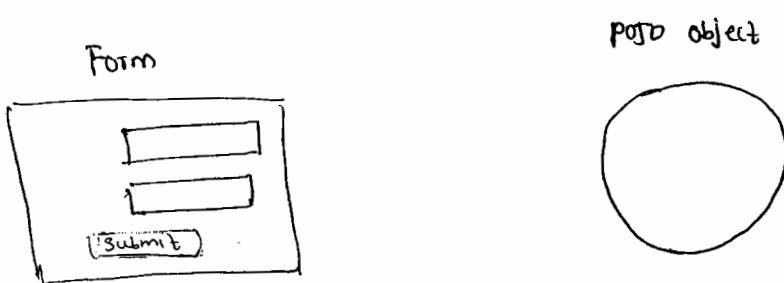
4-4-2015

* Using a POJO or command class or form backing class:

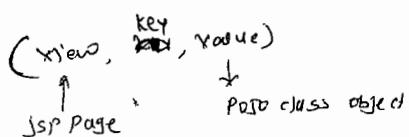
→ In Spring MVC, we use a POJO class for storing the form data submitted by end user there are two reasons for storing a form data in a POJO class object

- ① To reuse a form data in multiple methods or classes of the MVC application
- ② To apply validations on the input values.

URI → Uniform Resource Identifier



Key i.e. Command name to connect POJO class object & form design



"<http://www.springframework.org/tags/form>"

uri

→ When we want to store the form data in a POJO class object, before sending the form to the Browser, we should put a link bet'n form and the POJO class object.

→ a link can be put bet'n a POJO class object & a form by mapping key of a POJO class object as command name

in form design.

- Before a form is going to be sent to the browser, we need to create a POJO class object and we need to set ~~set~~^a key in controller bean.
 - If we want to do store a form data in POJO class object when it is submitted then we need to design the form using framework given ~~for~~ form tags, but not with ~~an~~ ordinary HTML form tags.
 - Spring framework has given tag library for a group of tags to design the forms.
 - When creating a JSP page we need to import the URI of tag library using taglib directive of JSP like the following
- ```
<%@taglib uri = "http://www.springframework.org/tags/form" %>
prefix = "form">
```
- ↓  
it can be  
anyname.

5-4-2015

applicable to all POJO class } JEE6 → Bean Validation API → given by Sun  
↓  
javax.validation.annotations

### Hibernate Validator API (Hibernate)

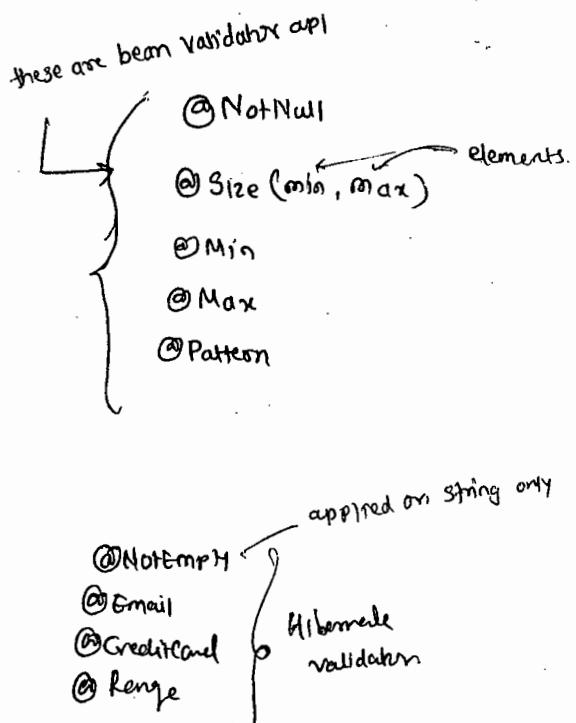
set of annotation parser class — contain some logic to read annotations  
necessary to read annotations

## Apply Validations on POJO

- As part of Java EE 6, Sun microsystem released bean Validation API with set of annotations for applying validation on properties of a POJO class.
- Bean Validation API contains annotations and the implementation of annotation (parsers) are provided by vendors.
- For ex: Hibernate has provided implementation for annotations for Bean Validation API and released in the form of Hibernate Validator API.
- In Spring framework, we use annotations of Bean Validation API with a Hibernate Validator implementation to apply the validations on POJO class.

Public class Login ⇒ POJO

{  
    @NotNull  
    Private String uname;  
        Property  
    @Size(min=5, max=8)  
    Private int pwd;  
        Property  
    @Min(18)  
    @Max(30)  
    Private int age;  
        Property



@NotEmpty

Private String uname;

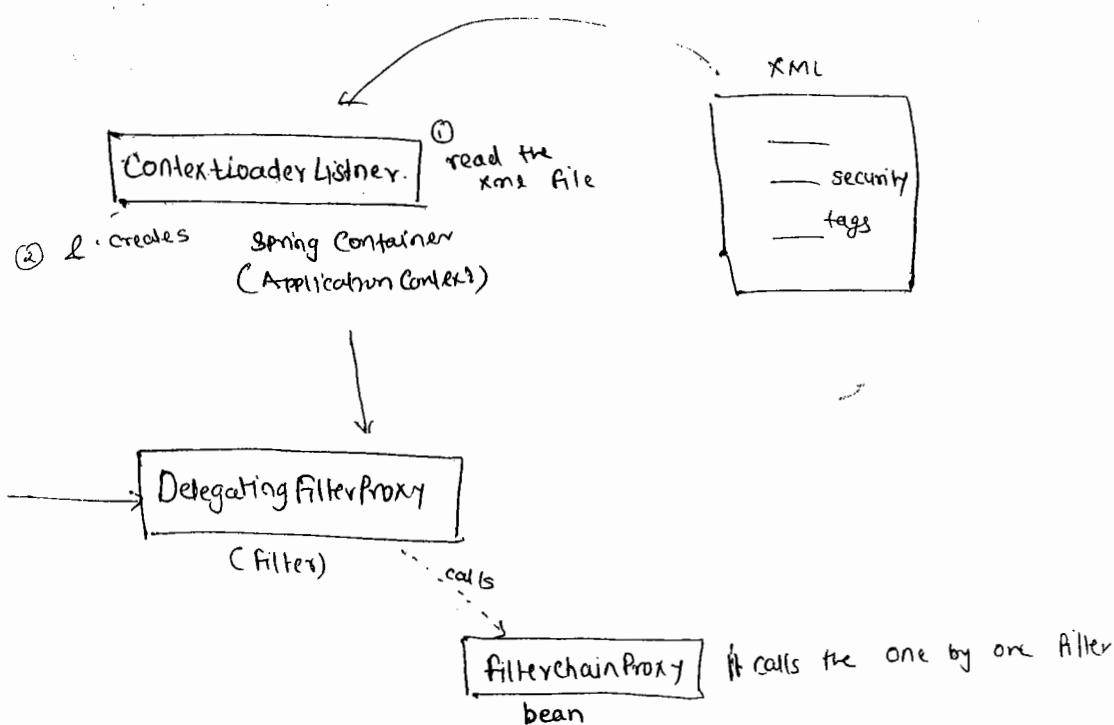
Sathyam Properties.

NotEmpty.Login.uname = NotEmpty.Key.uname = Value  
↳ error message  
Key of the object of POJO class

4-4-2015

## Providing Security to MVC application.

- an MVC app<sup>n</sup> contains multiple resources like HTML's, JSP's, Controller beans etc.
- In a web application security may not be required for each resource of the application, it means we make some resources as protected.
- In Spring framework to configure the security framework has provided a set of tags under security name space



- In Spring framework, the authentication logic to provide security in multiple filters but configuration of multiple filters in web.xml will increase the burden on developers.
- In order to reduce the developer burden, Spring API only has given predefined bean class FilterChainProxy and this class will

security related filters.

- So instead of writing all the filter configuration, we need to configure just ~~filter chain proxy~~ filterchainproxy bean in a Spring configuration file.
- To load the security configuration and to construct the a spring container object at application deployment time fw has provided a predefined Listener class called a ContextLoaderListener. this Listener class will read the spring configuration file from Context Parameter called ContextConfigLocation.
- When a request comes from browser, to trap the request and execute the security filter of the Spring, fw has given a filter DelegatingFilterProxy and we need to configure this filter in web.xml.

- In a web.xml file, apart from DispatcherServlet configuration, we also need to configure the following 3
  - ① Context Parameter.
  - ② Listener.
  - ③ Filter.

extension match → URL ends with \*  
ex → \*.form

directory match → Starts with dr  
ex → /dr

① <context-param>

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

<param-value> /WEB-INF / security-config.xml </param-value>

<context-param>

↑  
name can be anyone

② <listener>

<listener-class> org.springframework.web.context.ContextLoaderListener  
</listener-class>

<listener>

③ <filter>

<filter-name> springSecurityFilterChain </filter-name>

<filter-class> org.springframework.web.filter.DelegatingFilterProxy  
</filter-class>

</filter>

<filter-mapping>

<filter-name> springSecurityFilterChain </filter-name>

<url-pattern> /\* </url-pattern>

</filter-mapping>

Q. Why we are using same name as "springSecurityFilterChain"

Ans → When configuring the filter name (alias name) must be  
springSecurityFilterChain only because when a request comes then  
this filter class will read its alias name and it will search in  
the Spring Application Context container object for FilterChainProxy with this  
alias name as id

→ In Spring configuration, FilterChainProxy must be configured with  
the Spring Security SpringSecurityFilterChain as id the bean. So the  
link between DelegatingFilterProxy and FilterChainProxy bean is established  
using this alias name only.

5-4-2015

hasRole → security expression

↓  
to ~~use~~ use this

<security: http use-expression="true" />

one Role

↓  
group of Username  
& Password

<security: http basic>  
by default dialog box comes for login.

<security: form-login />

- ↳ by default gives login page
- ↳ also we can use our user defined login page.

repo.spring.io | released

↑  
select org  
↓  
spring framework  
↓  
security

<security: user-service> → In memory authentication.

and

<security: jdbc-user-service> database authentication.

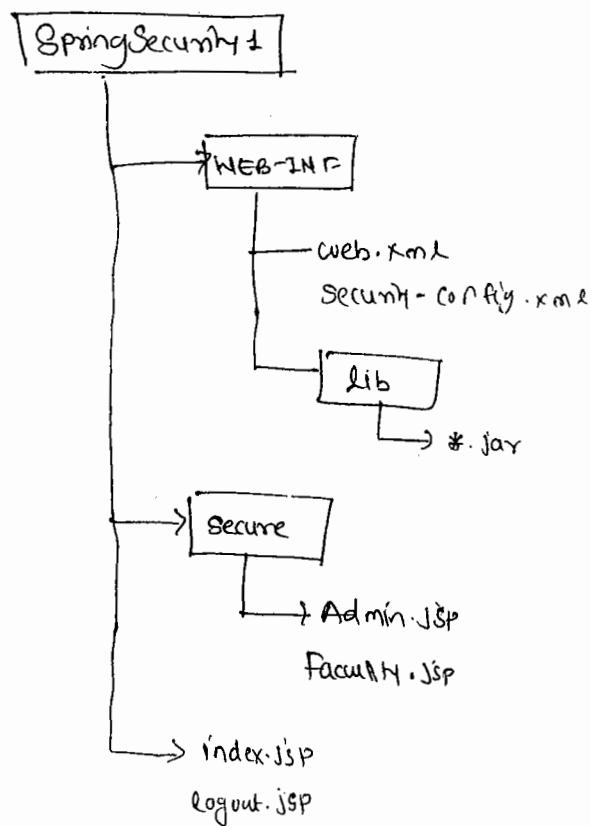
### Example

→ the following example contains two secured resources admin.jsp & faculty.jsp

→ we are going to configure the admin & faculty related username & password in spring xml file only. it means we are using In memory authentication

→ a problem with In memory authentication is, if we want to add new users or if we want to delete existing users then we need to reload the application again

→ in the following ex. we are not defining controller beans with logic. so we are not configure DispatcherServlet, HandlerMapping, ViewResolver in the application.



Refer. P.n. 12 for JEE Handout.

### List of jar

- aopalliance-1.0
- common-logging
- Spring-aop-4.0.5
- Spring-beans-4.0.5
- Spring-context-4.0.5
- Spring-context-support-4.0.5
- Spring-core
- Spring-expression
- Spring-security-config-3.2.7.RELEASE
- Spring-security-core-3.2.7.RELEASE
- Spring-security-web-3.2.7.RELEASE
- spring-tx-4.0.5.RELEASE
- Spring-web-4.0.5.RELEASE
- Spring-webmvc-4.0.5

### Note

We can download spring-security-api from the below website.

<http://repo.spring.io/release/org/springframework/security/spring-security/>  
→ download clicks.zip

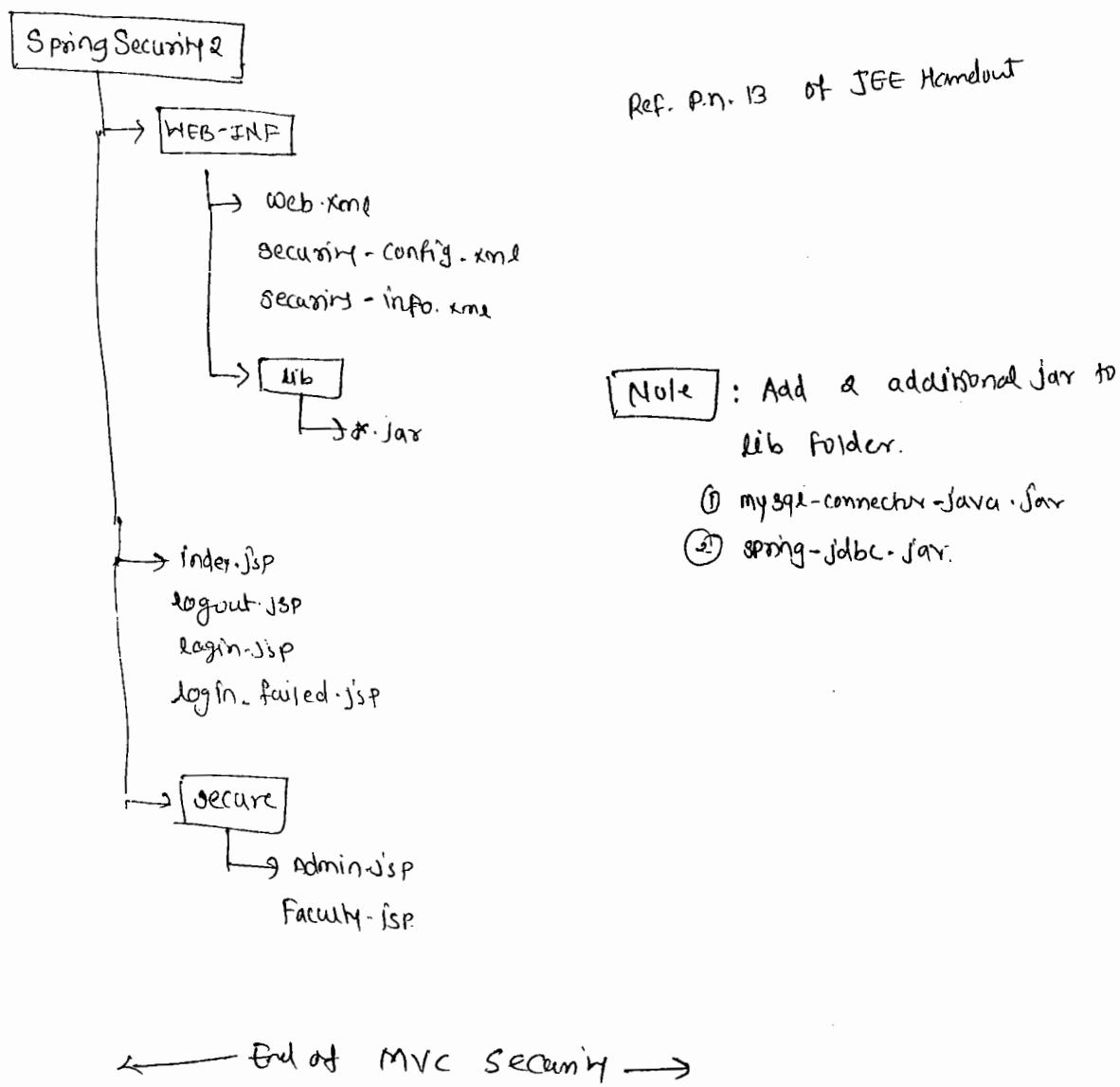
Example 2

Example 2

- this example is also for securing admin & faculty pages, but using JDBC authentication.
- for JDBC authentication we need to create two tables in MySQL like the following.

```
create table users (user_id int primary key, username varchar(10),
 password varchar(10), enabled tinyint);
```

```
create table user_roles (authority varchar(10), user_id int references users(user_id));
```



→ the list of annotations used for applying the validation are

- 1) `@NotNull`
  - 2) `@NotEmpty`
  - 3) `@Size(min=5, max=8)` → applied only on String
  - 4) `@Email`
  - 5) `@Min(value=18)`
  - 6) `@Max(value=25)`
  - 7) `@Pattern(regex="^ [A-Z][a-z]*")` → this pattern allows first letter as Uppercase & remaining lowercase.
  - 8) `@Range(min=10, max=20)` → applicable for numeric type (int, byte, short, long, float by me)
  - 9) `@CreditCardNumber`
  - 10) `@AssertTrue` { both are applicable for only boolean}
  - 11) `@AssertFalse`.
- Example      `@AssertTrue`  
                Private boolean b;

→ In spring framework we need to define the error messages called Validation in resource bundle.

→ in a resource bundle we need to follow the naming rule for a key to define the error message.

`annotationname. PojoObjectKey. propertyName = error message.`

for ex:

`NotEmpty.k1. uname = Username is required.`

→ to configure the resource bundle in a spring, we use `ResourceBundleMessageSource` bean in Spring configuration file.

```
<bean id = "messageResource" class = " ResourceBundleMessageSource >
 <Property name = "basename" value = "sathyam" />

 ↳ bundle name from
 </bean> (sathyam.properties)
```

7-4-2015

POJO

"annotations"

Bean Validation API  $\leftarrow$  used by any framework

Hibernate Validation API

<context:annotation-config>  $\rightarrow$  to enable the annotation used in Java class

<form:error>  $\rightarrow$  to display error message collecting from ResourceBundle

$\rightarrow$  When a form is submitted then the all values of the form are stored in a POJO class object by DispatcherServlet. When calling a controller bean method to tell the DispatcherServlet that inject the POJO class object to the method parameter, we use @ModelAttribute annotation.

$\rightarrow$  before injecting a POJO class object, to tell the DispatcherServlet that to exclude a validation, then we add @Valid annotation before @ModelAttribute annotation.

$\rightarrow$  while executing a validations, if any errors are occurred then those errors are stored BindingResult class object in the form of Error Objects.

$\rightarrow$  In a controller bean method, we need to take a Parameter of type BindingResult then DispatcherResult will pass BindingResult object to that method at the time of calling it.

## @Controller

```
public class LoginController
```

{

```
 @RequestMapping (value = "check", method = RequestMethod.POST)
```

```
 public ModelAndView processRequest (@Valid @ModelAttribute
 ("login") Login log, BindingResult errors)
```

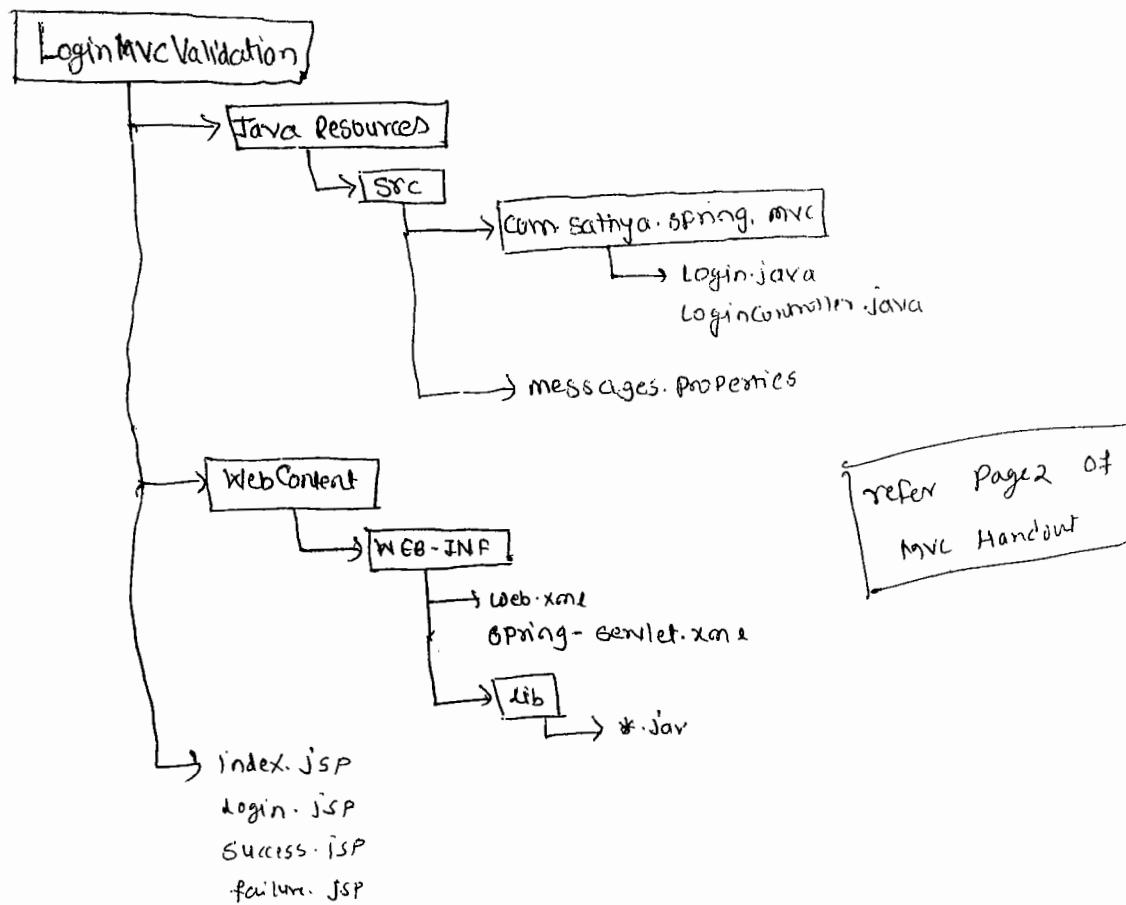
}

==

{

g

→ To show the error messages on browser we need to use a tag `<form:errors>` in the form design.



dependency

commons-logging

hibernate-validator-4.0.2.GA.jar

Servlet-api

slf4j-api-1.6.1.jar

Spring-aop

Spring-beans

Spring-context

Spring-context-support

Spring-core

Spring-expression

Spring-web

Spring-webmvc

newly added

Validation-api-1.0.0.GA.jar

8-4-2015

<form method="POST"

enctype="multipart/form-data" >

file upload "get" method  
not supported

by default browser add  
some encryption when  
sending data over server.

don't apply any encryption on file data.

Common-fileupload api

CommonsMultipartResolver

predefined  
class given  
by spring fwk

→ Internally uses  
File from request

→ Internally uses Commons-FileUpload  
api to read file from request object

→ convert file to multipart-file object  
→ return this object to DispatcherServlet

```
@Controller
```

```
Public class MyController
```

```
{
```

```
 RequestMapping
```

```
 p ModelAndView method (MultipartFile file)
```

```
{
```

```
 read & store object
```

```
}
```

```
}
```

## File Uploading in Spring MVC

- file uploading is nothing but transferring the content of the file from client system to servers.
- when a file uploaded then the uploaded file will be stored on a server either in HardDisk or Database.
- when designing a form per enduser to allow a enduser for selecting a file we need to provide a browse button. It is possible by using `<input type="file" />` tag of html.
- file uploading will not be supported by the http protocol get method, so we need to change method as POST in the form design.
- by default a browser <sup>will apply</sup> some encryption before it is sending data to the server but in file uploading we need to disable browser encryption. for this we need to add `enctype="multipart/form-data"` attribute to the form

→ by designing a form we need to make a few changes

① <form action = "xxx" method = "POST" ①

② enctype = "multipart/form-data" >

Select file : <input type = file > ③

<input type = submit >

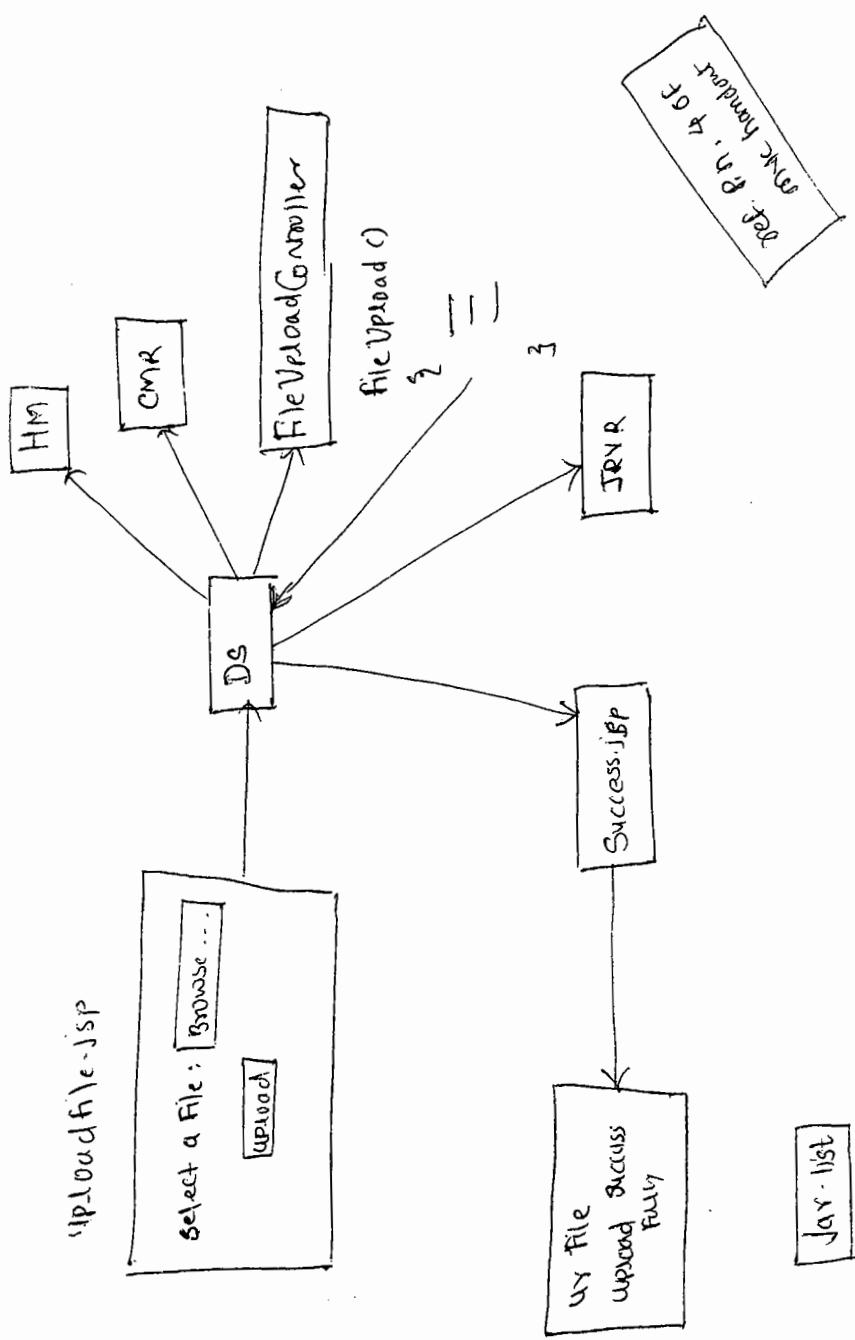
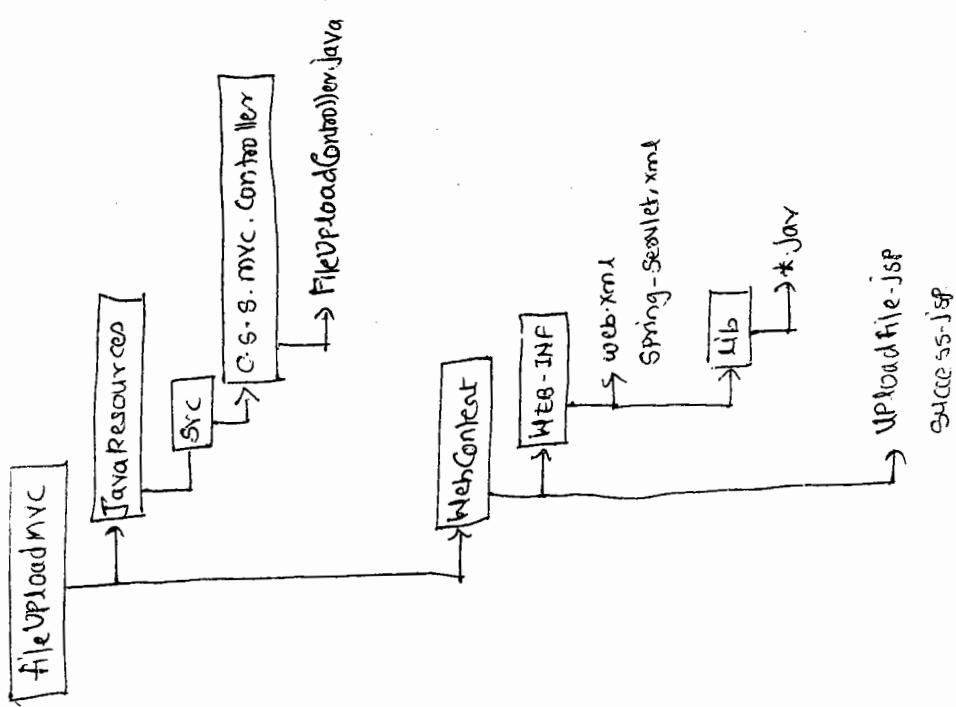
</form>

→ In spring framework a predefined class for CommonsMultipartResolver is given for converting an uploaded file into a MultipartFile Object

→ CommonsMultipartResolver uses commons-fileupload api of apache for converting an uploaded file into MultipartFile Object.

→ When calling a the method of Controller Bean DispatcherServlet will inject MultipartFile object to Parameter of the method and then we can define the logic for either storing the file on Server HardDisk or database in that method.

commons-fileupload-1.1.1 is depends on  
commons-io-1.1.jar



**Appearance**

```
{
 commons-fileupload-1.1.jar
 commons-io-1.1.jar
 commons-logging.jar
 jakarta-servlet.jsp-api-1.2.3.jar
 jakarta-servlet-api-1.2.3.jar
}
```

**Spring-context-support**

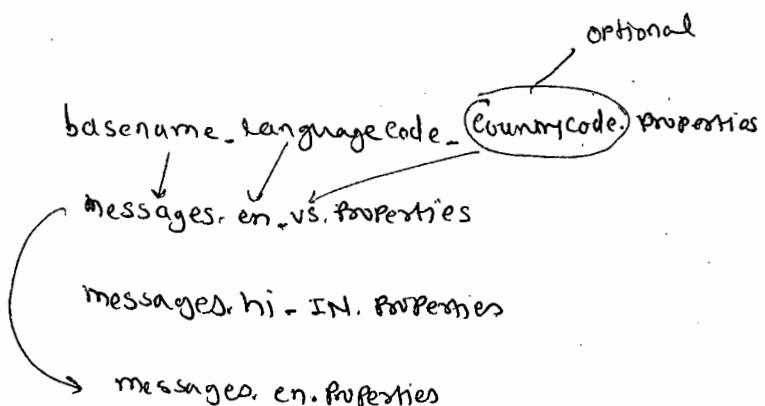
- Spring-core
- Spring-expression
- Spring-web
- SPRING-webmvc - 1.0.5

**Servlet API**

- Spring-AOP
- Spring-beans
- Spring-Context

14-4-2015

## I18N with MVC



→ I18N :- Internationalization.

→ I18N is a process of displaying the content on browser in a selected language by the end user.

→ by depending on the languages that are offered to the end user, we need to prepared that many number of Resource Bundle.

→ When Preparing the Resource Bundle the Bundle name should follow :

Syntax `basename_languagecode_Countrycode.properties`

→ In the above languageCode & countryCodes are ISO given codes. And the Countrycode is optional.

→ for ex: if we want to offer two languages English & Hindi to the end user we need to create two resource bundle with the following name.

- ① `messages_en_US.properties`
- ② `messages_hi_IN.properties`

OR

`messages_en.properties`  
`messages_hi.properties`

→ When defining the resource bundles, In all bundles keys must be same & there are values depending on the language.

→ IN I18N application, an application consists multiple resource bundle, but only 1 bundle should be loaded based on the language code for a selected language.

→ In order to load 1 bundle among multiple, we need to use the following 2 classes of ~~java.util~~ java.util package.

① Locale

② ResourceBundle

→ Locale is a class and ResourceBundle is an abstract class

```
String s = request.getParameter("lc");
```

```
Locale l = new Locale(s)
```

```
ResourceBundle rb = ResourceBundle.getBundle("messages", l);
```

→ If the value of requestParameter is en then the above code loads messages-en.properties file

→ When working with frameworks we no need of writing the above code in an application for loading a resource bundle. In place of code, we need to configured a predefined framework classes in configuration file.

→ In case of the Spring framework if we want to apply I18N support to MVC application then we need to configure the following.

→ 2 Predefined bean classes in spring configuration file.

① LocaleChangeInterceptor

② SessionLocalResourc

- LocaleChangeInterceptor class reads the value request parameter then creates a Locale object and then stores Locale object in Session object.
- SessionLocaleResolver reads Locale Object from Session, then loads a Resourcebundle for the given Locale.

```

<bean id = "change"
 class = "LocaleChangeInterceptor">

 <property name = "paramname"
 value = "lc" />

</bean>

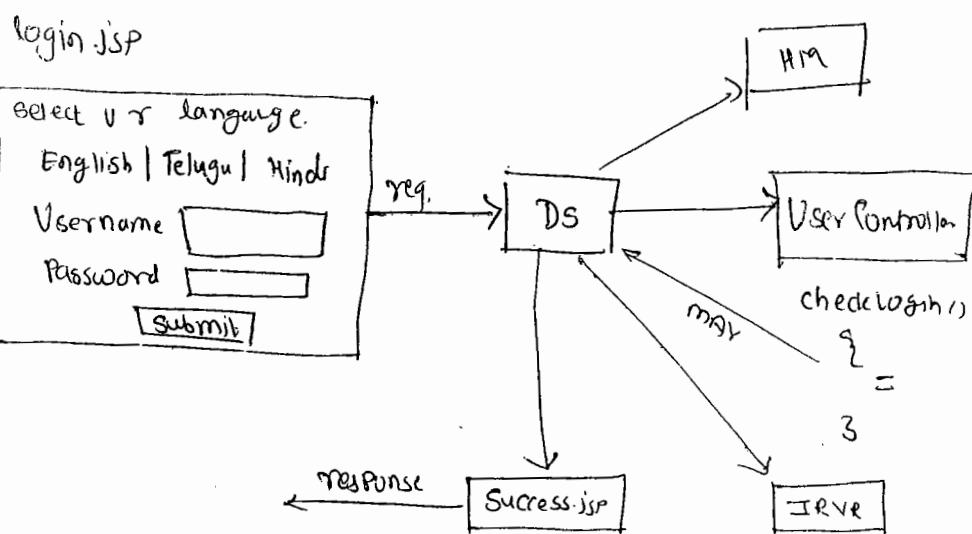
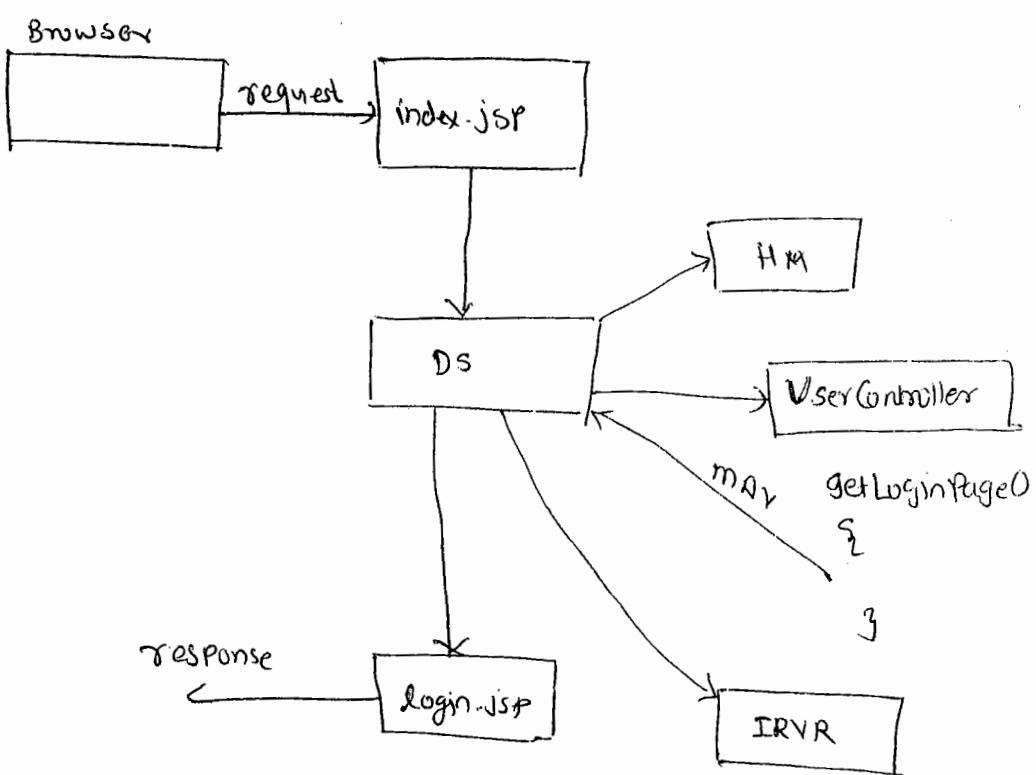
```

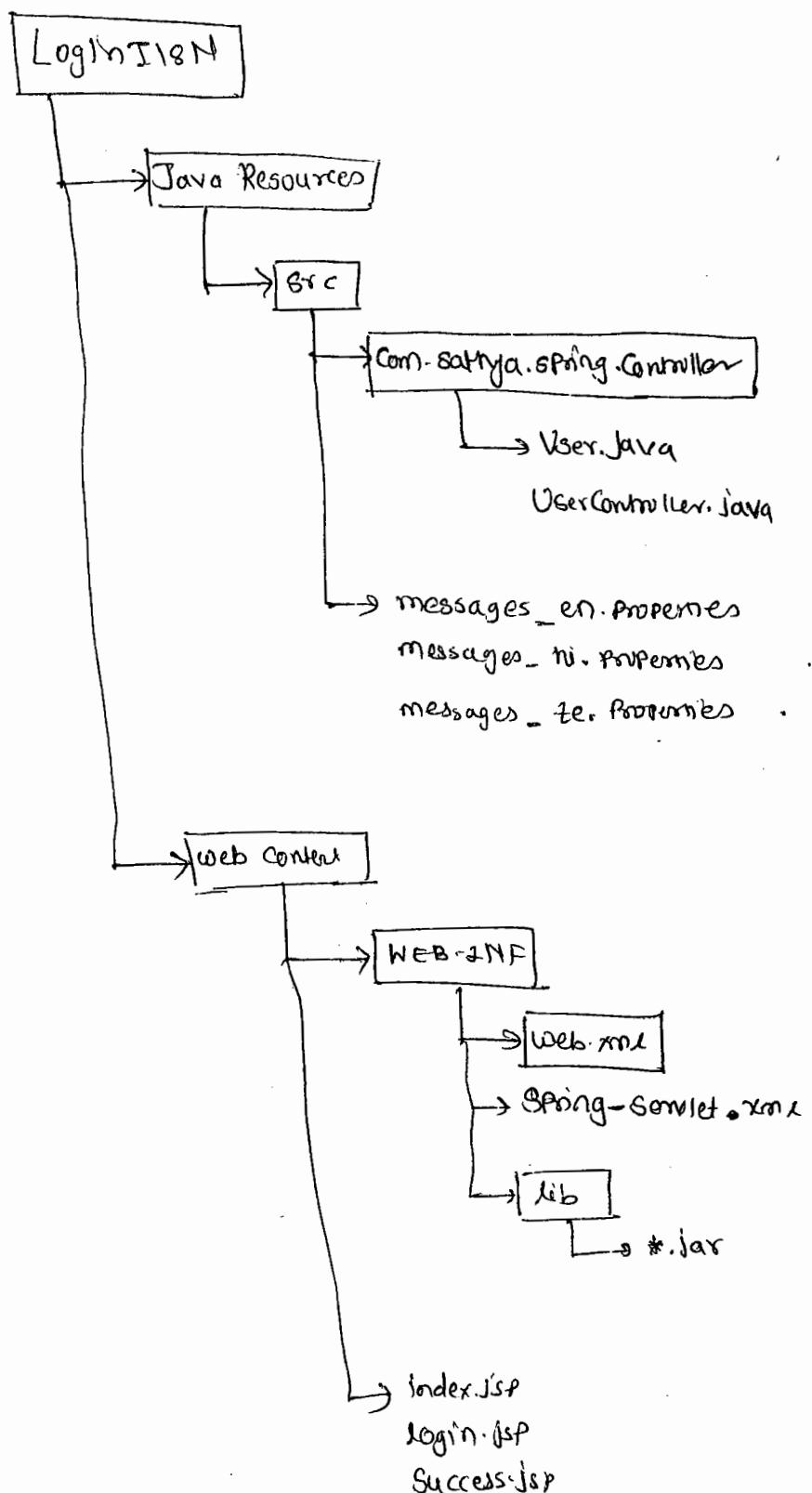
- here lc is a name request parameter through which a language code is sent from browser with request.

15-4-2015

### I18N Example

- In the following ex. we are creating a login app with validations. & we are offering login page in Hindi, English, & Telugu, by applying I18N.





→ **Dependence on:**

- commons-logging
- hibernate-validator-4.0.2.GA
- slf4j-api-1.6.1.jar
- servlet-api.jar
- spring-aop
- spring-beans

**depends on:**

Spring Context  
 spring-context-support  
 spring-core  
 spring-expression  
 spring-web  
 spring-webmvc  
 validation-api

16-7-2015

### User.java

```
public class User
{
 @NotEmpty
 private String uname;
 @NotEmpty
 private String pwd;
 // setters & getters
}
```

### UserController.java

@Controller

```
public class UserController
```

{

    @RequestMapping (value = "/login", method = RequestMethod.GET)

```
 public ModelAndView getLoginPage()
```

{

        return new ModelAndView("login", "command", new User());

}

    @RequestMapping (value = "/check", method = RequestMethod.POST)

```
 public ModelAndView checklogin (@Valid @ModelAttribute("command")
```

        @ModelAttribute("command") User user, BindingResult br)

```

 }

 if (br.hasErrors())
 {
 return new ModelAndView("login");
 }
 else
 {
 return new ModelAndView("success");
 }
}

```

### Spring-servlet.xml

```
<context:component-scan base-package = "com.sathya.spring.Controller" />
```

```
<!-- handler mapping -->
```

```
<bean class = "org.springframework.web.servlet.handler.DefaultHandlerMapping" />
```

```
<!-- View Resolver -->
```

```
<bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver" />
```

```
 <Property name = "prefix" value = "/" />
```

```
 <Property name = "suffix" value = ".jsp" />
```

```
</beans>
```

```
<bean id = "messageSource" class = "org.springframework.context.support.ResourceBundleMessageSource"
```

```
 <Property name = "basename" value = "messages" />
```

```
</beans>
```

```
<mvc:annotation-driven />
```

## Index.jsp

```
<jsp:forward page = "login.form"> </jsp:forward>
```

## Login.jsp

```
<%@ taglib uri = "http://www.springframework.org/tags/form" %>
 prefix = "form" %>
```

```
<%@ taglib uri = "http://www.springframework.org/tags" %>
 prefix = "spring" %>
```

Select V + language ?

```
 English
 || Telugu
 || Hindi
```

```



```

```
<center>
```

```
<form:form action = "check.form" method = "post" commandName
 = "command">
```

```
<spring:message code = "label.username"/> : <form:input Path =
 "uname" /> <form:errors Path = "uname" /> <spring:message code =
 "NotEmpty.command.uname" />
```

```
<spring:message code = "label.password" /> "NotEmpty.command.pwd"
 /> </form:errors>

```

```
<spring:message code = "label.password" /> : <form:password Path = "pwd" />
```

```
<form:errors Path = "pwd" /> <spring:message code = "NotEmpty.command
 .pwd" /> </form:errors>

```

```
<input type = "submit" value = "submit" />
```

```
</form:form>
```

```
</center>
```

## messages\_en.properties

```
messages_en.properties
```

```
label.username = Username
```

```
label.password = Password
```

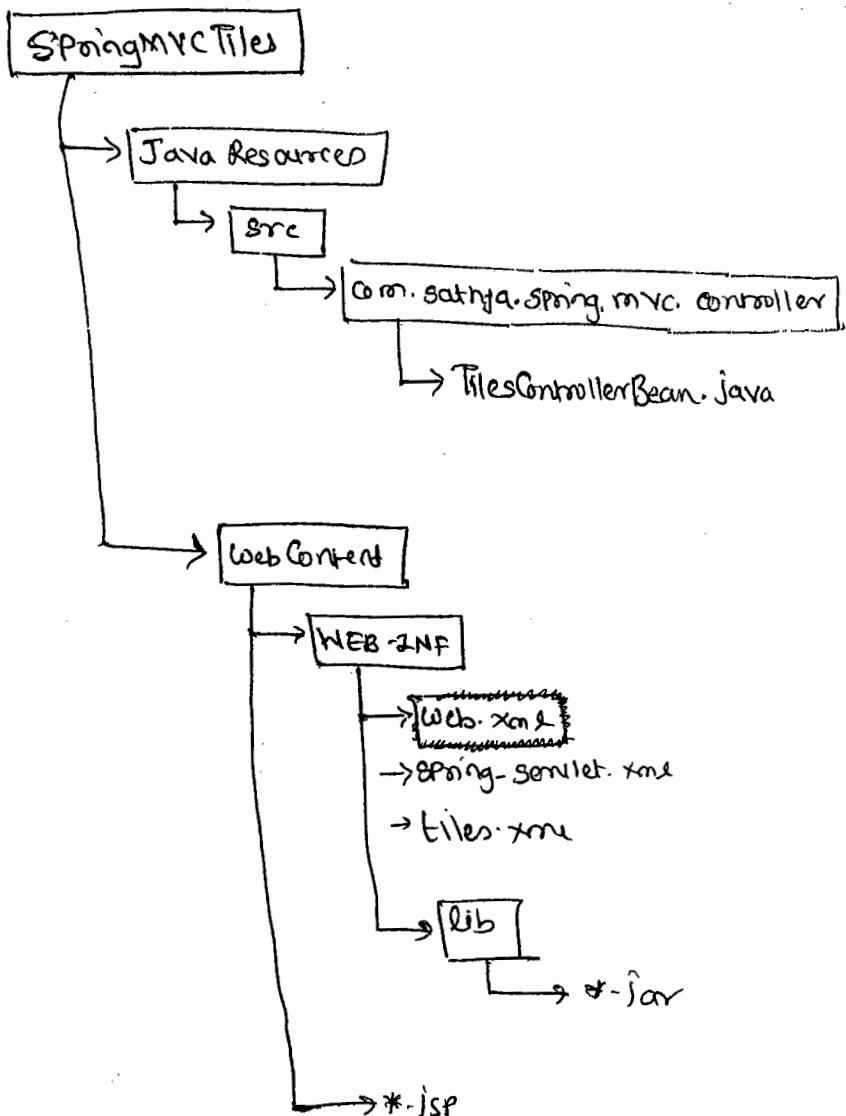
```
NotEmpty.command.uname = Username is required
```

```
NotEmpty.command.pwd = Password is required
```

Note → In the remaining 2 bundles also, we need put some keys but the values are html text and it can be generated or collected collected from below website

<http://vikku.info/inian-language-unicode-converter>

18-4-2010



Refer. Page - 9  
MVC Handout.

## Jar file list

ApacheCommons-4.0

Commons-beanutils-1.8.0

Commons-digester - 2.0 .jar

Commons-logging

Javaee.servlet.jsp.jstl-1.2.1

Javaee.ServlU.jsp.jstl-api-1.2.1

SIP4J-api-1.6.1

Spring-aop - 4.1.2 .RELEASE

Spring-beans - 4.1.2 .RELEASE

Spring-context - 4.1.2 .RELEASE

Spring-context-support - 4.1.2 .RELEASE

Spring-core - 4.1.2 .RELEASE

spring-expression - 4.1.2 .RELEASE

spring-web - 4.1.2 .RELEASE

Spring-webmvC - 4.1.2 .REL

tiles-api - 3.0.5

tiles-autotag - core-runtime - 1.1.0 .jar

tiles-compat - 3.0.5 .jar

tiles - core - 3.0.5 .jar

tiles-el - 3.0.5

tiles-extras - 3.0.5

tiles-jsp - 3.0.5 .jar

tiles-request-api - 1.0.6

tiles-request.jsp - 1.0.6

tiles-request.servlet - 1.0.6

tiles-request-Servl  
wildcard - 1.0.6

tiles-servllet - 3.0.5

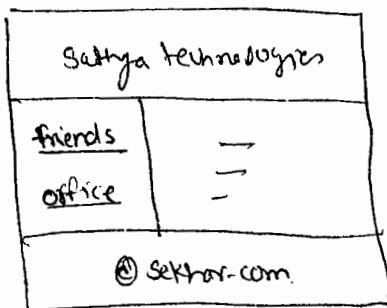
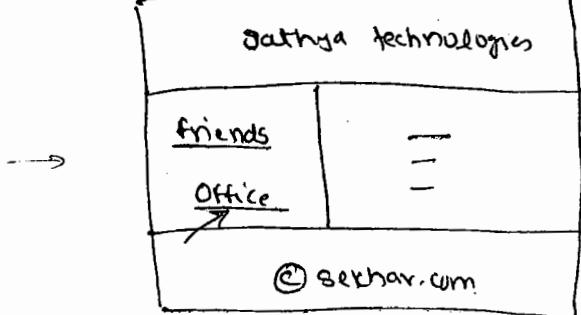
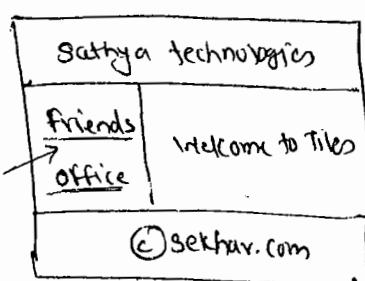
tiles-tempfile - 3.0.5

```

<bean id="xxx" class="TilesConfigurer">
 <property name="definitions">
 <list>
 <value>/WEB-INF/tiles.xml</value>
 </list>
 </property>
</bean>

```

- When working with tiles application, we need to configure ~~tiles~~ TilesViewResolver in spring configuration file. This ViewResolver will read appropriate view from tiles container and returns a tilesView object to the DispatcherServlet.
- the following example contains 3 composite views.
- to create a composite we are using Spring MVC with tiles flow of apache.
- the composite views are following a common structure with 4 regions.



## tiles.xml

```
<tiles-definitions>
 <definition name = "def1" template = "layout.jsp">
 -- -
 </definition>
 <definition name = "def2" extends = "def1">
 -- -
 </definition>
 <definition name = "def3" extends = "def1">
 -- -
 </definition>
 </tiles-definitions>
```

- if a large number of definitions are needed then we can define multiple tiles configuration file.

## Spring beans to work with tiles:

- Spring fw has given a bean class for loading the one or more tiles configuration file called TilesConfigurer.
- TilesConfigurer internally uses tiles API for reading the definitions of tiles configuration file and finally prepared the tiles container object.
- ~~TilesConfigurer~~ TilesConfigurer class has a ~~property~~ "definition" & we need to configure one or more TilesConfiguration file as a list of Value

### \* Creating a layout page

- a layout page is to define the structure of the composite view pages into a separate page.
- In order to work with tiles framework a 1<sup>st</sup> step is creating layout page
- a layout page divides a view into a multiple regions & assigning a name for each region
- while creating a layout page to create a different regions we use <table> tags of HTML.
- to assign a name to each region we use a tiles tag of apache called <tiles: insertAttribute>
- In a layout page, to use a tiles tag of apache we need to import a tag library uri into our jsp page using a taglib directive.

```
<%@taglib uri = "http://tiles.apache.org/tags-tiles"
 prefix = "tiles"%>
```

### \* tiles Configuration file

- the composite views of a web application which follows a layout are configured in a tiles configuration file.
- a tiles-configuration file contains one or more definition where each definition indicates a composite view of the application if multiple composite views contains common content in some regions then to get reusability we can create a base definition and we can extend it into other definitions of configuration file.

<mvc: interceptors>

```
<bean id = "localeChangeInterceptor"
 class = "org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
 <Property name = "ParamName" value = "p1"/>

```

</bean>

</mvc: interceptors>

```
<bean id = "localeResolver"
 class = "org.springframework.web.servlet.i18n.SessionLocaleResolver"/>
```

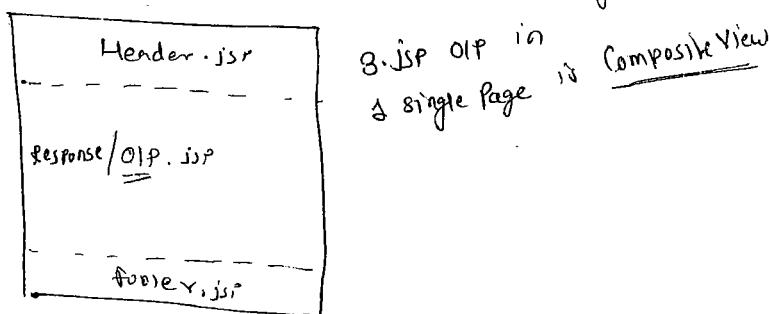
</beans>

## Web.xml

Copy Paste

## Spring MVC with Tiles

Response of multiple page or  
single page.



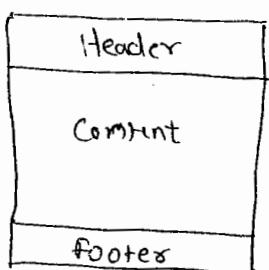
```
<jsp:include page = "header.jsp"/>
```

==  
==

```
<jsp:include page = "footer.jsp"/>
```

- In web application, if response <sup>to the</sup> ~~on~~ browser is constructed by a single JSP Page then it is called Simple view. If response is constructed by including response of other JSP Pages including them it is called Composite view.
- To create Composite views, we include the response of other JSP Pages with the help of <jsp:include> action tag
- When an web application contain a large number of Composite View Pages then we use the support of Tiles framework given by Apache.
- By without using support of tiles framework <sup>also</sup> we can develop composite views, but when change is required in the layout of Pages then we need to open and modify each Composite Page.
- For eg: ① We are creating a web application with Composite View Pages, by displaying a Header on top, a footer on bottom, and content in a middle of the page.

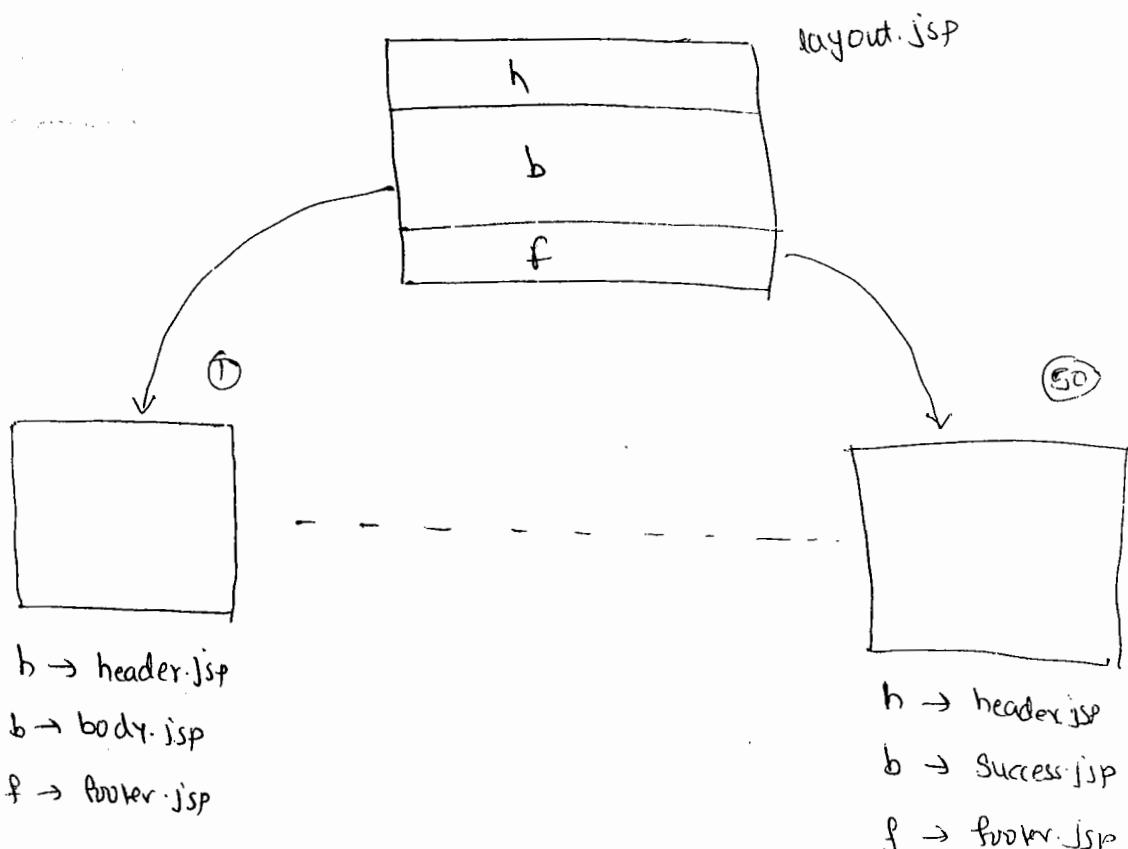
②



- ② If our web application containing 50 composite views and if we want to change a look after composite view like a Header as footer & footer as Header. Then we need to open & modify all the 50 composite view pages. It will be burden on developer and a it is a time consuming process.

③ In order to reduce the above problem apache introduce tiles fw.

- Tiles fw of apache can be used by any web application framework which wants to create a composite view pages in a web application
- In spring-mvc applications also, we use Tiles fw of apache
- In tiles programming 1<sup>st</sup> we need to create a layout page to define the structure of the composite views.
- When creating composite views, we insert the structure to all the composite views.
- the benefit of using tiles fw is when we want to change the structure or look of the composite views then we can just only just modified & making the layout page automatically the changes are affected on all the composite views.

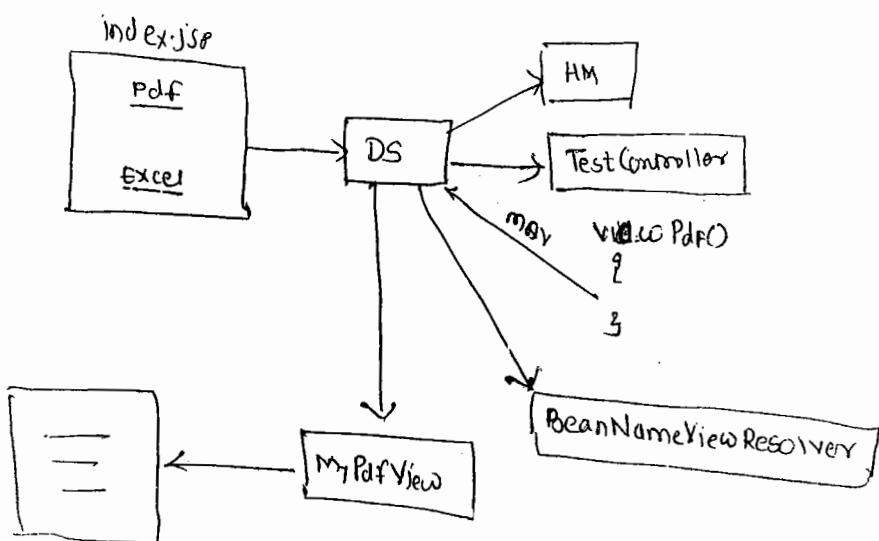


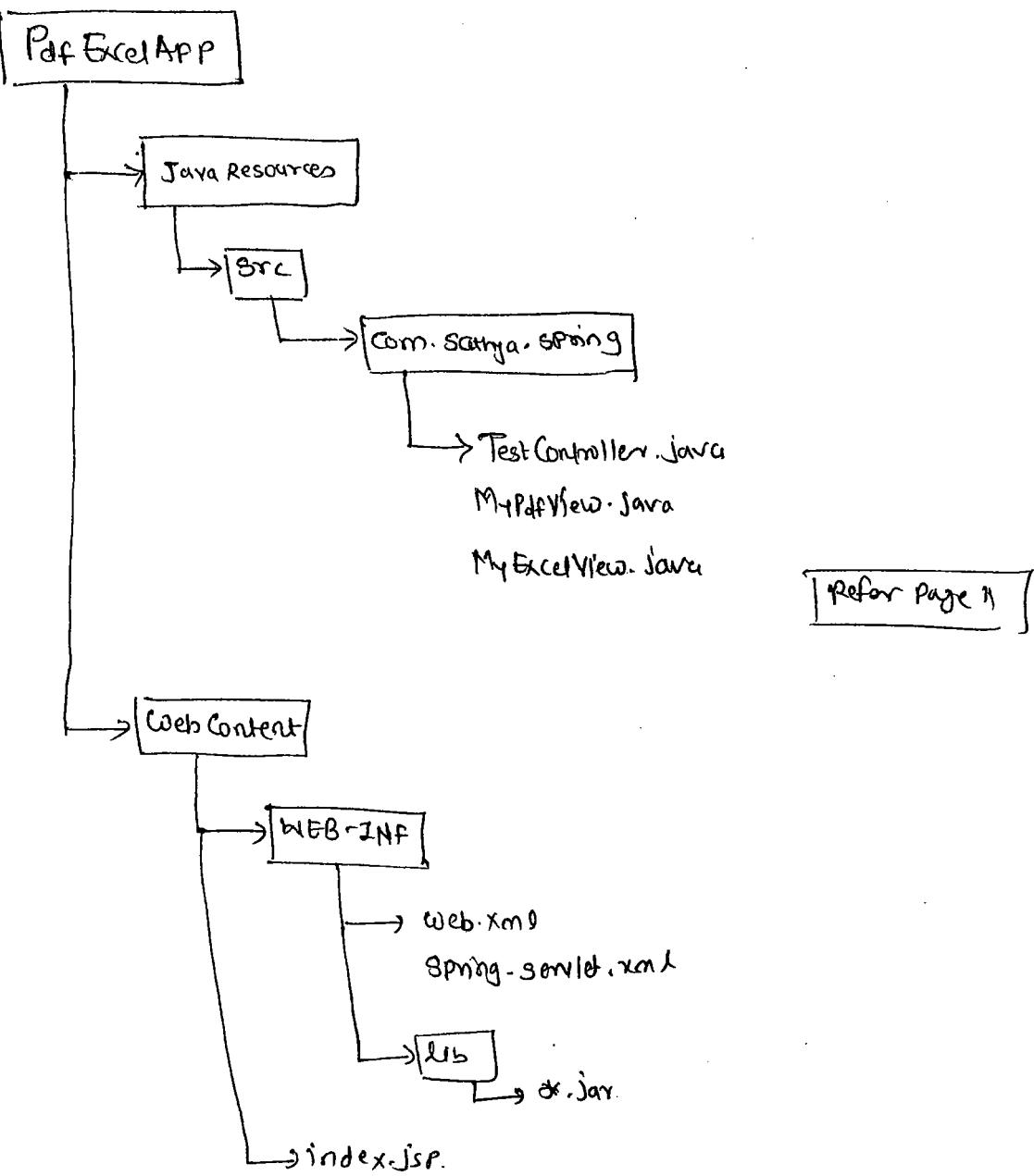
⇒ suppose if want to modify header as footer & footer as header  
in a 50 composite pages then we need to just open the layout page &  
then we need to modify region name h as f & f as h.  
automatically all the 50 <sup>composite</sup> pages are modified.

## Generating Response in Pdf & Excel formats.

- In Spring MVC, to generate response like a Pdf Page OR an Excel Page then there are 2 options
- Option ① in JSP Page, we can set MIME type as PDF or Excel & we can define the logic
- Option ② we can create a Bean class and we can extend from abstract base classes ~~provided~~ given by the framework.
- Spring framework has provided 2 abstract base classes called ~~Abstract pdf view and Abstract~~ AbstractPdfView & AbstractExcelView.
- AbstractPdfView is an abstract class with one abstract method buildPdfDocument(), so we need to override this abstract method
- AbstractExcelView is also an abstract class with one abstract method buildExcelDocument(). so we need to override this abstract method.

### Example





### JAR

- ① cupalliance-1.0
- ② commons-logging
- ③ iText-2.1.7  
newly added
- ④ poi-3.9
- ⑤ Servlet-api
- ⑥ spring aop
- ⑦ spring beans
- ⑧ spring context
- ⑨ spring Context-support
- ⑩ spring-core
- ⑪ spring-expression

- ⑫ spring web
- ⑬ spring webMVC.

```
public class MyExcelView extends AbstractExcelView
{
 protected void buildExcelDocument(Map m, HSSFWorkbook wb, HttpServletRequest req,
 HttpServletResponse res) throws Exception {
 Sheet sheet = wb.createSheet("Sheet1");

 Row r1 = sheet.createRow(0);
 Cell c1 = r1.createCell(0);
 c1.setCellValue("Company Name");

 Cell c2 = r1.createCell(1);
 c2.setCellValue("Website");

 Row r2 = sheet.createRow(1);
 Cell c3 = r2.createCell(0);
 c3.setCellValue("Oracle");
 Cell c4 = r2.createCell(1);
 c4.setCellValue("http://www.oracle.com");

 Row r3 = sheet.createRow(2);
 Cell c5 = r3.createCell(0);
 c5.setCellValue("IBM");
 Cell c6 = r3.createCell(1);
 c6.setCellValue("http://www.ibm.com");
 }
}
```

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

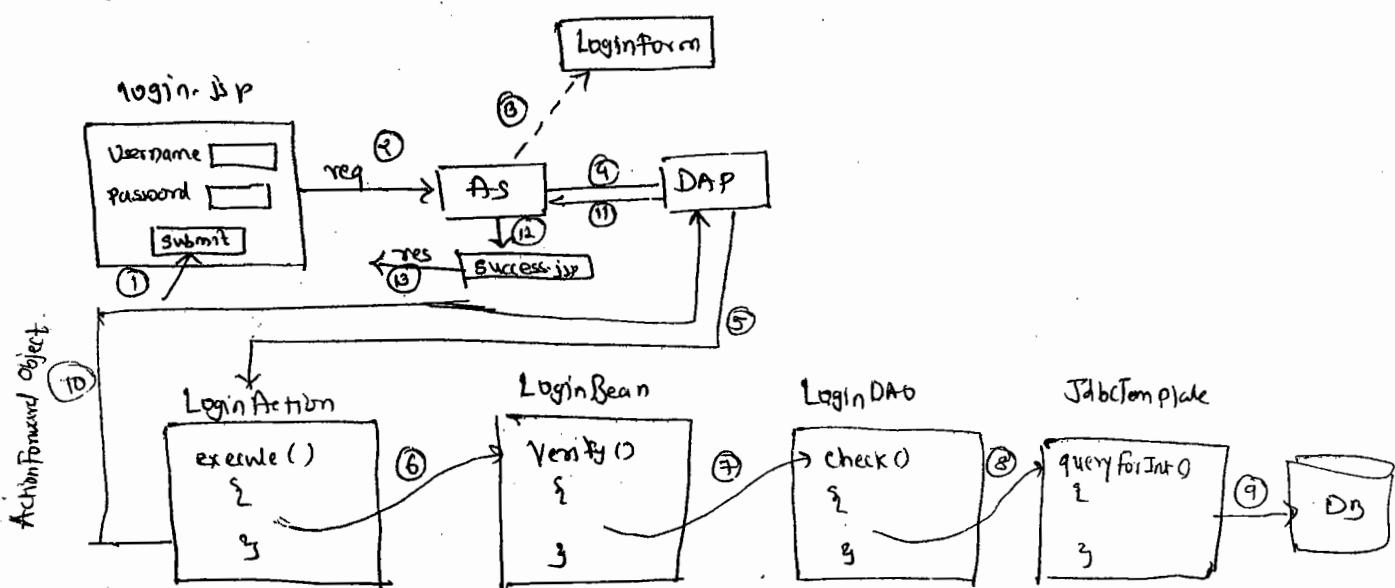
C

C

C

Q. Plugin for integrating struts & spring

Context Loader Plugin given by ~~struts~~ Spring fwk



Q. Diff. bet' beanfactory & Applicationfactory Context container

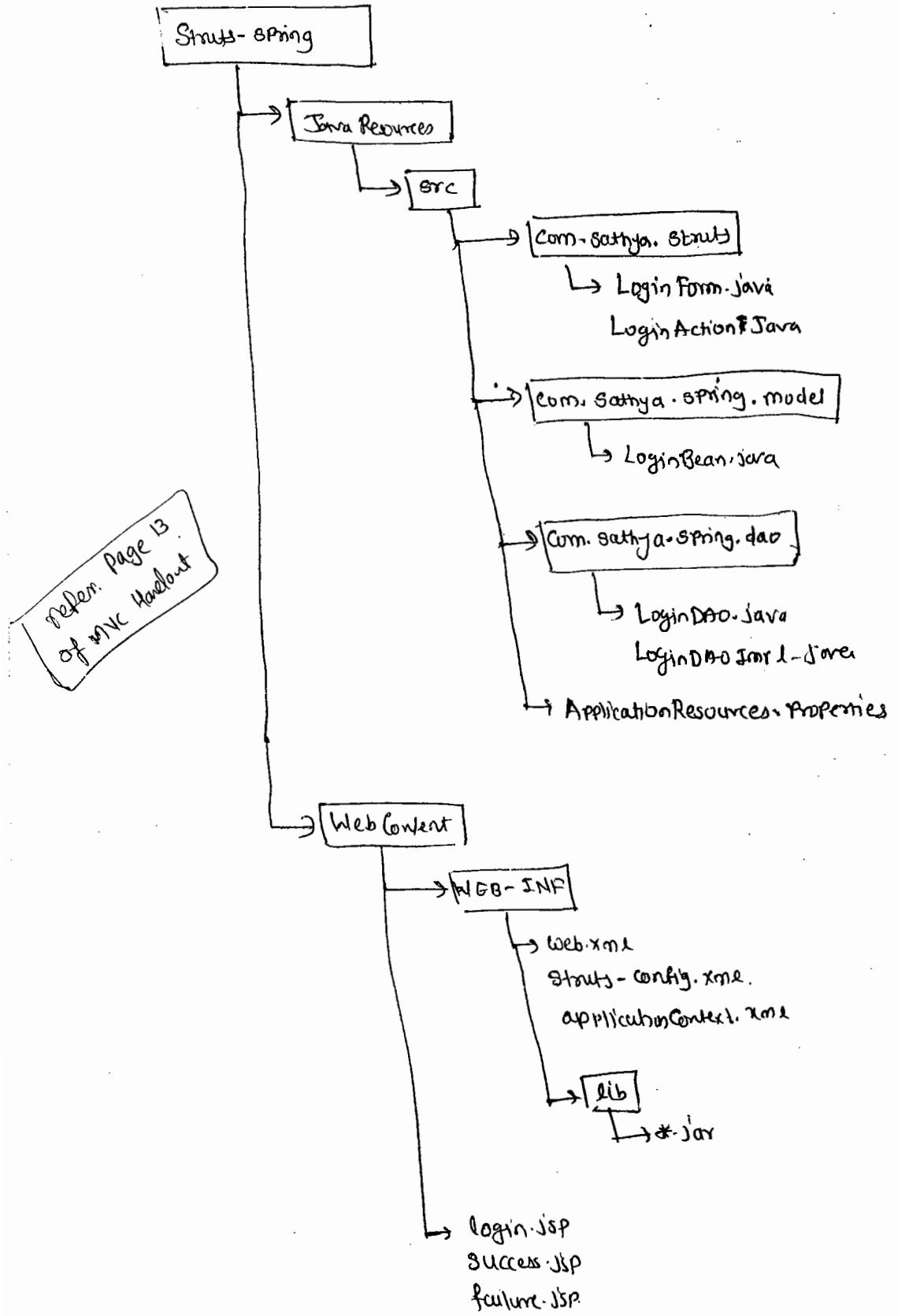
### Example

→ In the following example, we are creating login appn for integrating struts & spring with database.

→ LoginAction class of struts will call the LoginBean class of spring.

→ LoginBean class of calls a DAO calls & then it ~~calls~~ <sup>calls</sup> JdbcTemplate to execute the sql command / query on database.

↳ Next Page.



## Jar

- ① antlr-2.7.2
- ② commons-beanutils-1.7.0
- ③ commons-chain-1.1
- ④ commons-digester-1.8
- ⑤ commons-logging-1.0.4
- ⑥ commons-validator-1.3.1
- ⑦ jdbc14.jar
- ⑧ ojb-2.08
- ⑨ spring-beans-4.1.2.RELEASE
- ⑩ spring-context-4.1.2.RELEASE
- ⑪ spring-context-support-4.1.2.RELEASE
- ⑫ spring-core-4.1.2
- ⑬ spring-expression
- ⑭ spring-jdbc
- ⑮ spring-struts-3.2.4.RELEASE.jar → not spring fw download separately
- ⑯ spring-tx
- ⑰ spring-web
- ⑱ spring-webmvc
- ⑲ struts-core-1.3.8
- ⑳ struts-taglib-1.3.8

## \* ResourceBundleViewResolver

- If URL's of views are defined in a ResourceBundle then to read the URL and to return a view object, ResourceBundleViewResolver is used.
- In a ResourceBundle, for each view we need to define two keys  

viewname.class	&	viewname.url
----------------	---	--------------
- ResourceBundleViewResolver loads a ResourceBundle and returns a URL in the form View class object.
- In Spring configuration file we need to configure ResourceBundle name as a Value for baseName Property.

### In Spring-servlet.xml:

```
<bean id="viewResolver"
 class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
 <Property name="basename"
 value="spring-view"/>
</bean>
```

In InternalResourceViewResolver view name  
& page name (.jsp) must be same, but  
in ResourceBundleViewResolver not necessary  
may be different

### Spring-views.properties

success.class = org.springframework.web.servlet.view.JstlView

success.url = /ok.jsp

failure.class = org.springframework.web.servlet.view.JstlView

failure.url = /sorry.jsp.

### \* SimpleUrlHandlerMapping

→ this URLHandlerMapping class will identify appropriate controller bean like the following

- ① It reads the url pattern of Request.
- ② It searches for a key in a mappings property matched with url pattern.
- ③ It reads the value of key and then it will search for bean whose id is matched with that value.
- ④ Finally returns an object of controller bean to the DispatcherServlet.

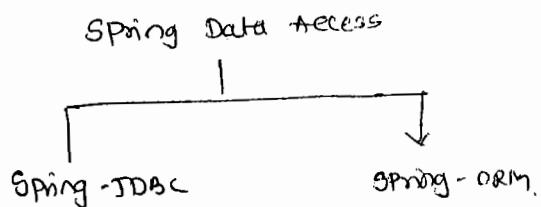
→ with SimpleUrlHandlerMapping, we need to configure mappings property. this Mappings is a collection of type ~~Properties~~. Property.

for ex:

```
<bean id="xx" class="SimpleUrlHandlerMapping">
 <Property name="mappings">
 <Props>
 <Prop key="/login.form">id1</Prop>
 <Prop key="/register.form">id2</Prop>
 </Props>
 <Property>
 </bean>
```

# Spring Data Access Module

9-4-2015



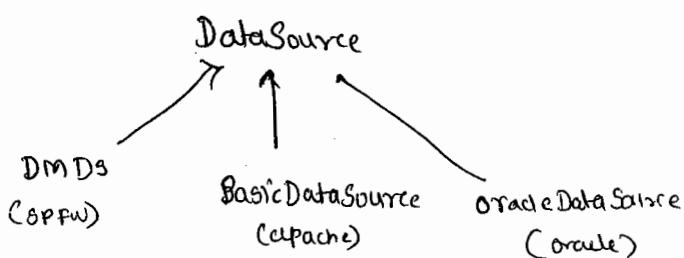
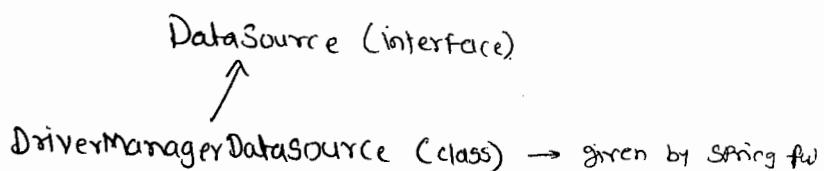
## Spring - JDBC

→ Spring-JDBC is an abstraction layer on top of JDBC technology when directly working with JDBC technology ~~use~~ the following 3 problems identified by the programmers

- ① Boiler plate code (repetitive code)
- ② checked exceptions.
- ③ Memory leak

→ In order to reduce the above problems Spring framework has provided a layer on top JDBC & named it as Spring-JDBC

JdbcTemplate → given by fw



## ~~JDBC~~ JdbcTemplate

- Spring framework has provided a central class for executing "CURD" operation on ~~database~~ called JdbcTemplate.
- this JdbcTemplate class depends on a DataSource object.
- JdbcTemplate class has provided convenient methods for executing SQL commands on database.
- JdbcTemplate class will internally take care about Boiler Plate code i.e required for executing the SQL commands.
- In Spring, the checked exception of JDBC are translated as unchecked exception of Spring framework so a programmer is no need handle the exception by adding try & catch block.
- When working with JdbcTemplate, a programmer only concentrates on sending SQL queries to database and processing results of the query.

### Public class JdbcTemplate

- In JdbcTemplate class a constructor and also a setter method are defined for injecting a DataSource object. so Spring configuration file we can configure JdbcTemplate class like the following

```
<bean id = "jt" class = "JdbcTemplate">
 <constructor-arg ref = "ds" />
</bean>
```

[OR]

```

<bean id="jt" class="JdbcTemplate">
 <property name="dataSource"
 ref="ds"/>
</bean>

```

```

public class JdbcTemplate {
 private DataSource dataSource;
 public JdbcTemplate (DataSource ds) {
 this.ds = ds;
 }
 public void setDs (DataSource ds) {
 this.ds = ds;
 }
}

```

constructor

better method

↳ dataSource

- DataSource is an interface that framework has provided one implementation class of DataSource interface called a DriverManagerDataSource.
- In Spring Configuration file, we can configure either framework provided implementation class or Vendor provided implementation class. for example : BasicDataSource is an implementation class of DataSource interface of an apache

→ So we can configured BasicDataSource class also in spring configuration file

- execute & update are for non-select

- query method for select operation on database.

① execute(): ".create / alter / truncate / drop"  
(DDL command)

```
Scanner s = new Scanner(System.in);
```

→ sid  
→ sname  
→ marks

```
Object arr[] = {sid, sname, marks};
```

```
int i = jdbcTemplate.update("insert into student values (?, ?, ?)",
 arr);
OR
```

```
int i = jdbcTemplate.update("insert into student values (?, ?, ?)",
 sid, sname, marks);
Varargs
```

DDL Commands index parameter (?) not allowed

① execute(): this method is only for executing DDL Commands  
(create, alter, truncate, drop)

```
jdbcTemplate.execute("create table xyz (id numbers), name Varchar(10));");
```

Note: In DDL command, "index parameter (?)" are not allowed.

② Update(); this method is used to execute DML commands (insert, update, delete)

→ this method returns integer value which indicates the number of rows updated or affected.

→ we can use Update method in 3 ways

① update (command) } → static command

② update (command, Object []) }

③ update (command, Object ..., ~~varargs~~) } dynamic command

④ update (command, Object ...) }

↓  
varargs

### Example: 1

```
int i = jt.update ("insert into student values (101,'A',500);") ;
```

### Example 2

```
Scanner s = new Scanner (System.in);
```

```
sop ("enter sid");
```

```
int sid = s.nextInt ();
```

```
sop ("enter sname");
```

```
String sname = s.next ();
```

```
sop ("enter marks");
```

```
int marks = s.nextInt ();
```

```
Object [] arr = { sid, sname, marks};
```

```
int i = jt.update ("insert into student values (?, ?, ?)", arr);
```

Ex. 3

```
int i = jt.update ("insert into Student values (?, ?, ?)",
 sid, sname, marks);
```

④

~~queryForMap();~~

10-4-2015

⑤

queryForMap

- It is used for reading a single row from a database
- this method returns a Map, which contains columns name as key and gotten column values as value.

→ to make the query as dynamic we can use index parameter & we can set the values by passing an Object[] or Varargs

- a) queryForMap (command)
- b) queryForMap (Command, Object [])
- c) queryForMap (Command, Object ...)

Ex:

```
Map m = jt.queryForMap ("select * from emp where empno=7788");
for Map object it is not possible to use Iterator. so convert it
Set s = m.entrySet();
```

```
Iterator it = s.iterator();
```

```
while (it.hasNext())
```

```
{
```

```
 Map.Entry me = (Map.Entry) it.next();
```

```
 System.out.println (me.getKey () + " " + me.getValue());
```

```
}
```

④ `queryForList();`

→ this Method is used to read the Multiple rows from database

~~List list =~~ ↗

→ Internally, spring ~~gives~~ stores each row data in a Map & then stores Map Objects to a list & then returns list object

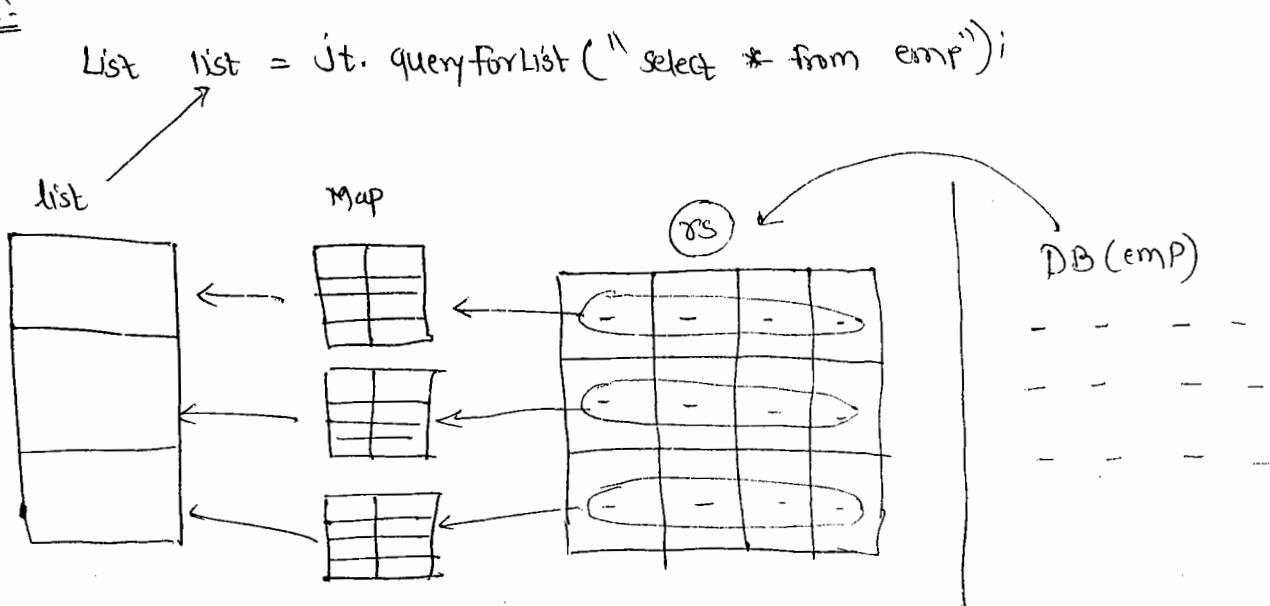
→ to make the query as dynamic, we can pass index parameters and the values can be set by creating an Object {} or varargs.

a) `queryForList(Command)`

b) `queryForList(Command, Object[])`

c) `queryForList(Command, Object...)` ↗  
called Varargs

Ex:

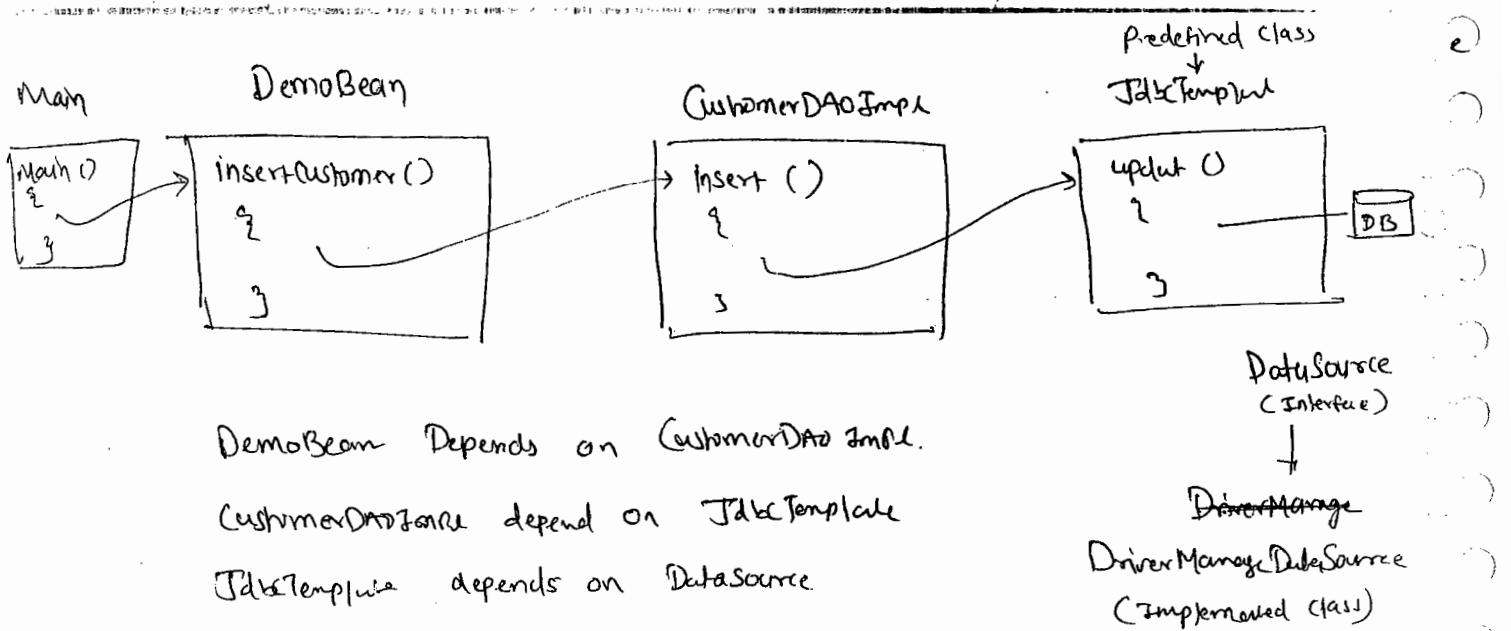


`Iterator it = list.iterator();`

`while (it.hasNext())`

{  
  `Map m = (Map) it.next();`

`Set s = m.entrySet();`



XMLE

business class Configuration

`id = db class DemoBean`

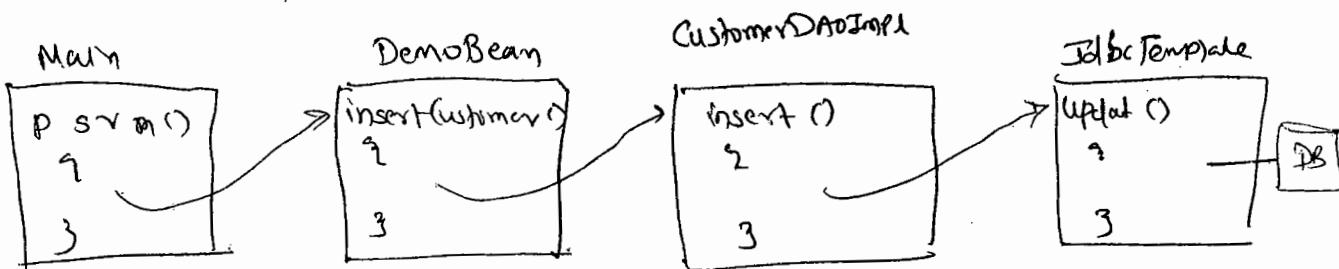
JAR

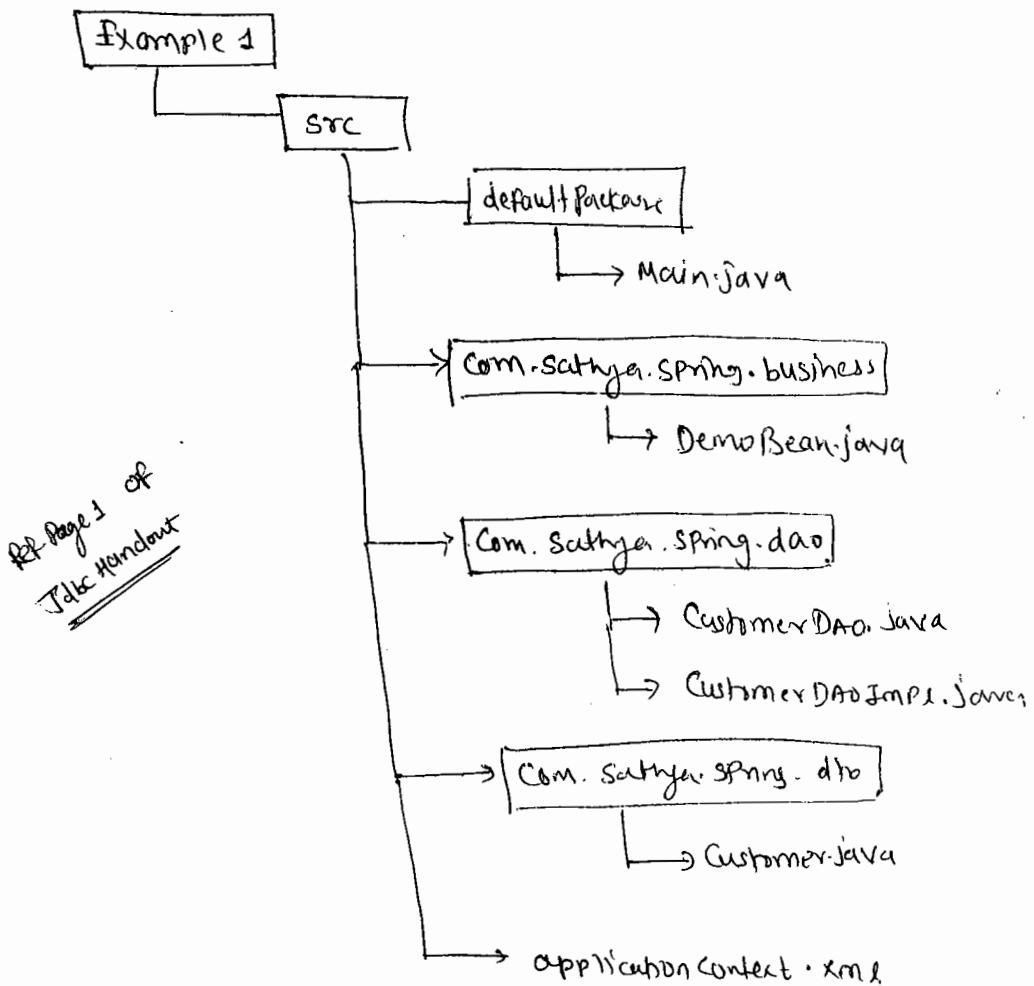
Spring jdbc depends on  
`spring-tx`

11-4-2015

Example 1

this example is to insert the values by calling `update()` method of `JdbcTemplate`





## List of jar

newly added

### **Spring tx**

Spring beans

Spring-context

Spring-context-support

Spring-core

Spring-expression

newly added

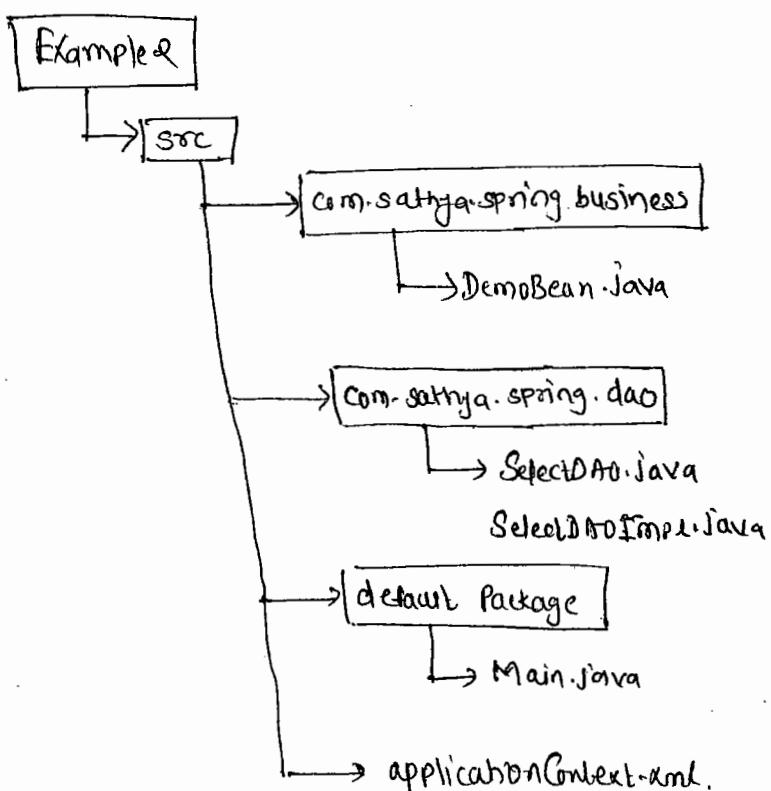
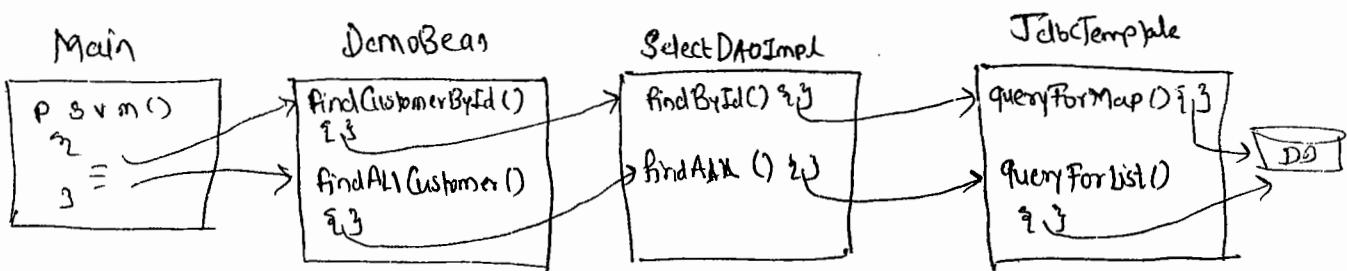
### **Spring-jdbc**

Commons-logging

Mysql-connector / JDBC

## Example 2

The following example is for reading customers by calling query methods of JdbcTemplate.



## List of Jars

- common-logging
- mysql-connector
- spring-tx
- spring-beans
- spring-context
- spring-context-support
- spring-core
- spring-expression

Spring-JDBC

tomcat-dbcp

Spring-mop → depends on  
lenny-aop-alliance)  
 application

~~queryForInt()~~  
~~queryForLong()~~

} ⇒ deprecated in 4.x.  
 (means if we called this Method automatically  
 they ~~strikes~~ strikes).

### Note

queryForInt() and queryForLong() methods are deprecated from Spring 4.x instead we use queryForObject.

Q What is callback ~~interface~~ in JdbcTemplate.

RowMapper is a callback interface.

\* Query with RowMapper.

→ In Spring-Jdbc the framework has provided a feature to the programmer that a programmer can pass some code into predefined methods of JdbcTemplate class.

→ To define the code a programmer should implement a class from callback interfaces.

→ RowMapper is one of the callback interfaces that a programmer can implement this interface when a programmer wants to convert each row of a ResultSet object into a POJO class object.

→ Public class EmployeeMapper implements RowMapper

{

② Override

public Object mapRow(ResultSet rs, int index) throws SQLException

$\hookrightarrow$   
 Employee e = new Employee();  
 e.setXX(rs.getXX());  
 return e;

$\hookrightarrow$  throws exception because  
 we are using ResultSet object

We pass an object of EmployeeMapper to the query() method of JdbcTemplate. that query() methods calls ~~mapRow()~~ mapRow() method of the class for each row of Resultset.

Predefined method in JdbcTemplate.  
query (String sql, RM rm)

{

Opening connect  
create statement

Rs rs = stmt. executeQuery(sql);

for each row of the  
Resultset it will call  
MapRow method

while (rs.next())

Object o = rm. mapRow (rs, i);

arraylist.add(o);

return arraylist;

↳ contains Pojo class object.

}

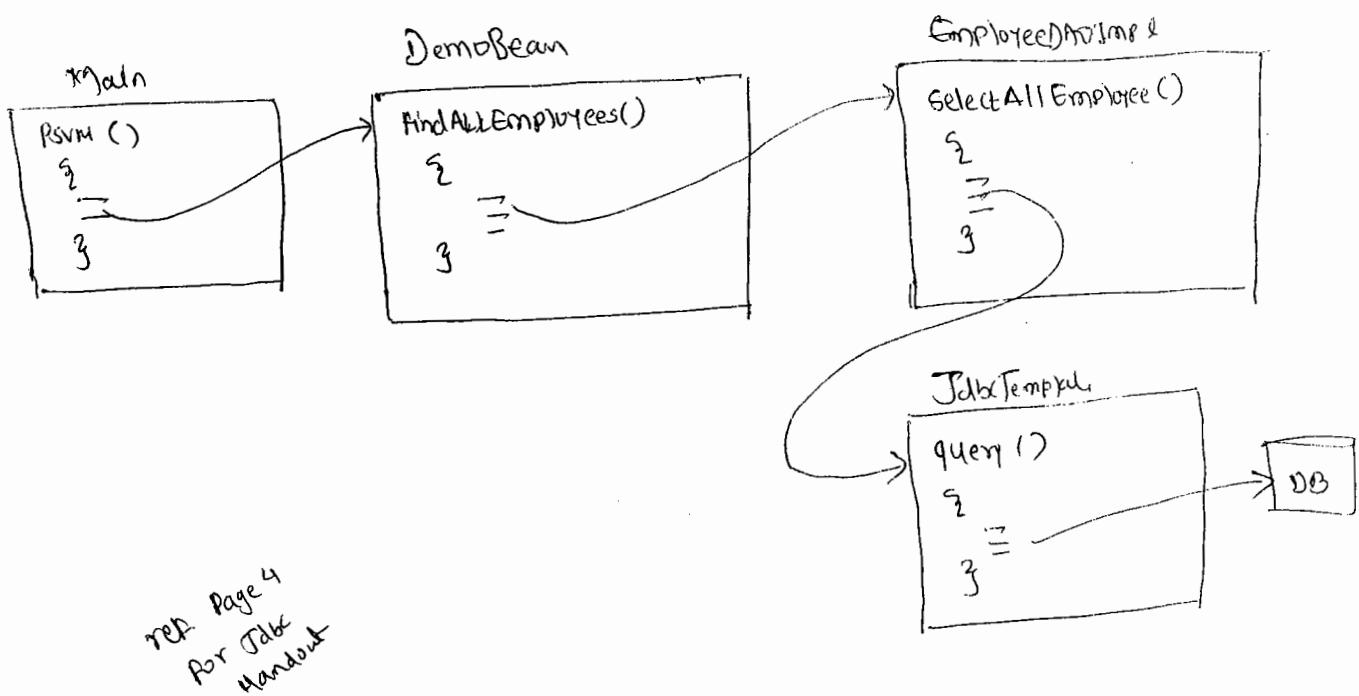
→ query() method JdbcTemplate stores all the objects in  
a list and finally returns list object.

List list = jt.query ("select \* from emp", new EmployeeMapper());

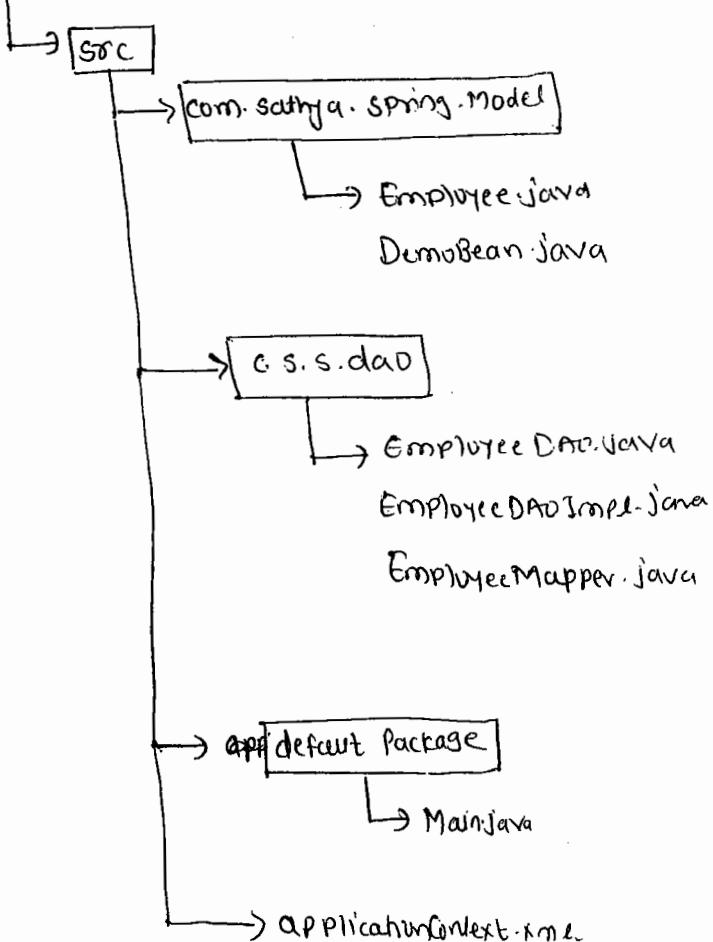
### Example 3

→ In the following Example we are converting each row of  
Resultset ~~obj~~ into an Object of Employee Pojo class, by  
implementing our class from a callback interface called RowMapper.

### Example 3



### Example 3



FR  
V2MP

## Q. RowMapper & ResultSetExtractor diff

Query with ResultSetExtractor

public class

### \* Query with ResultSetExtractor \*

- ResultSetExtractor is another callback Interface like a RowMapper.
- Query() method of JdbcTemplate calls <sup>class</sup> extractData() method of ResultSetExtractor one time by passing complete ResultSet object
- In extractData() method a programmer can decide which type of Collection object is required and also a programmer can decide what rows of ResultSet need to be converted ~~into~~ to the Pojo class objects.



→ the difference bet' RowMapper & ResultSetExtractor is in a RowMapper collection is a list and there is a guarantee that for each row of ResultSet a pojo class object is created. but in ResultSetExtractor a Collection type can be decided by programmer and a programmer can decide for what rows of ResultSet, pojo class object ~~is~~ are required.

Public class EmployeeExtractor implements ResultSetExtractor

{

⑥ override

Public Object extractData(ResultSet rs) throws SQLException

{

Set s = new HashSet();

int count = 1;

While (rs.next())

{

If (count == 1 || count == 10)

{

Employee e = new Employee();

e.setXXX(rs.getXXX());

s.add(e);

}  
Count++;

return s;

}

→ according to the above code collection type is decided by the programmer as Set and only 1<sup>st</sup> and 10<sup>th</sup> rows are converted into POJO class object.

→ we can call query() method of JdbcTemplate like the following

```
Object o = jt.query("select * from emp", new EmployeeExtractor());
```

```
Set s = (Set)o;
```

```
Iterator it = s.iterator();
```

```
While (it.hasNext())
```

{}

```
Employee e = (Employee) it.next();
System.out.println(e.getName());
}
```

## batch processing in Spring-Jdbc:

means multiple trips  
converted into single trips

Statement → ① addBatch  
② executeBatch()  
general command combine  
called batch

- select command not allowed  
in batch.

JdbcTemplate → batchUpdate()

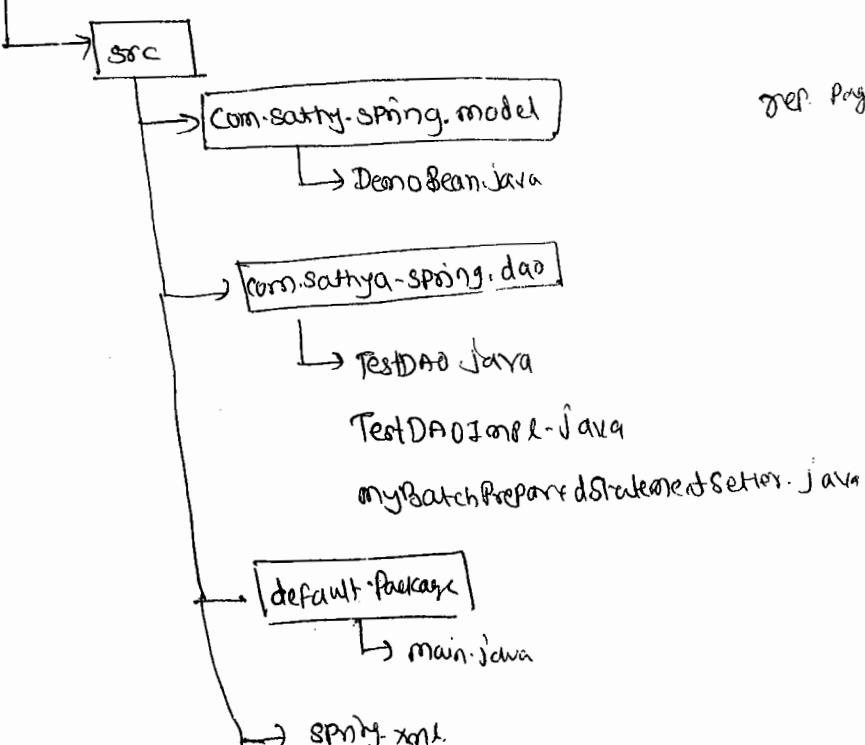
- batch processing means sending multiple commands to database in one trip.
- In JdbcTemplate class, batchUpdate() method is given for batch processing.
- batch processing can be done by static command & dynamic commands.
- for static command, first we need to store the static command in a String [] array, and then we need to pass that String [] array to as a parameter to batchUpdate(). method.

```
String str [] = {"insert into student values (1,'a',400)",
 "update emp set sal=9000 where empno=7788",
 "delete from emp where empno = 7901"};
```

```
int i = jt.batchUpdate(str);
```

- for batch Processing using a dynamic query, to define the batch size & and also to set the values to the query for multiple times we need to implement our class from a callback interface called `BatchPreparedStatementSetter`.
- `BatchPreparedStatementSetter` Interface has two abstract method
  - ① `getBatchSize()`
  - ② `setValues()`
- first `getBatchSize()` method called & based on the int value returned `setValues()` method called that many number of time.

### Example 4



ref. Page. 8 of Jdbc Handout

Note: → before executing the application, we need to create table in database like the following.

`create table Test(id number(15) primary key, name varchar(10));`

## Calling a procedure or function

- In Spring JDBC also we need to create logic using Callable Statement
- for calling a procedure or function of database
- In JDBC Template class, there is a method called call() for calling procedure or function, it returns a map object which contains the output values return by the procedure or function
- In order to create a Callable Statement object, we need to implement our class from callback interface CallableStatementCreator.

for ex:

public class MyCallableStatementCreator implements CallableStatementCreator

{

    @Override

    public CallableStatement createCallableStatement(Connection con) throws

SQLException

{

    CallableStatement cstmt = con.prepareCall ("{ ? }");

    ----

    return cstmt;

}

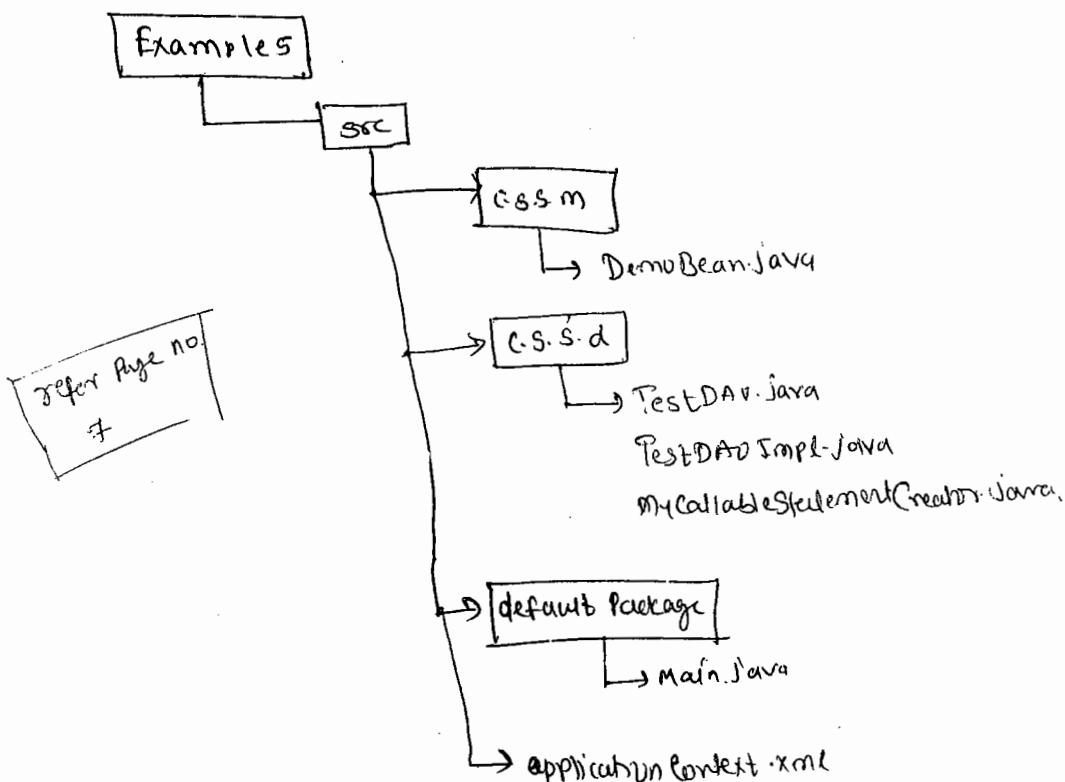
}

- for each parameter used in ~~Repos~~ Callable Statement CallableStatement we need to ~~use~~ create a SQL Parameter class object for representing it. we create SQL Out Parameter class object for representing OutParameter of an CallableStatement.

- we need to add SQL Parameter object to the array list & then we need to pass that array list as parameter to the

### callC() method of JdbcTemplate.

- the following Example is going to call a function of database using JdbcTemplate
- A function of database reads emp as an Employee Number & returns bonus as an output.



- because executing the above we need create a function in oracle like the following

SQL> ed f1 → enter it will open f1 file

create or replace function fun1 (eno number)

return number is

bonus number (5);

temp number (0);

begin

select sal into temp from emp where

empno = eno;

```

if temp between 1 and 2000 then
 bonus := temp * 0.1;
else
 bonus := temp * 0.2;
end if;
return bonus;
end;

```

SQL > @f1  
IS 1  
function created.

## Spring - ORM

- this spring - orm is an abstraction layer for orm tools like a hibernate, jdo, & ibatis
- this spring-orm layer is introduced for the following 3 reasons.
  - ① to reduce Boiler Plate Code used when working with orm tools.
  - ② to provide common exception hierarchy to make exception handling as easy.
  - ③ To use on orm tool as part global transaction.
- spring framework has provided common exceptions Hierarchy for the multiple data access technology. it means spring throws some exceptions for any data access <sup>technology</sup> like jdbc or hibernate or jdo - etc.
- the orm tools we used popularly is hibernate and the Spring-orm has provided a template class for reducing Boiler Plate of code of hibernate as "HibernateTemplate".

HibernateTemplate starts with Session  
depends on  
LocalSessionFactoryBean → Session factory object

- HibernateTemplate class need a Sessionfactory and the it will be produced by local SessionfactoryBean
- LocalSessionFactoryBean is a factoryBean which produces Sessionfactory of Hibernate.
- Spring container injects Sessionfactory produced by LocalSessionFactoryBean to the HibernateTemplate

```
<bean id = "ht" class = "HibernateTemplate">
 <property name = "sessionfactory" ref = "lsfb"/>
</bean>

<bean id = "lsfb" class = "LocalSessionFactoryBean">
 -- -- --
</bean>
```



# Spring-Services Module

21-4-2015

## ④ Scheduling service

24-4-2015

### @Schedule Annotation

Public class xxx  
fixed Rate / fixed Delay / cron

{  
    @Scheduled (xxxx)  
    Public void dowork()

}  
=

3

@Scheduled (fixedRate = 5000) → for every 5 sec dowork() method called

Public void dowork()

{  
    =

@Scheduled (fixedRate = 5000)

Public void dowork()

{  
    Thread.sleep(1000);  
    =

previous Task then 5 sec Delay next task  
complete

- Scheduling is a service used for automatic a job.
- When we want to execute a job (work) automatically by without explicitly calling then we need to apply this scheduling
- A scheduler will automatically invoke a method in which the logic is defined, at a specific point of time.
- Ex In a mobile we can scheduled an alarm and the it will executed automatically according to the schedule.
- In spring framework to define the schedule we have a method level annotation @Scheduled.
- @Scheduled annotation can have one of the following three elements
  - ① fixedRate
  - ② fixedDelay
  - ③ cron

→ FixedRate is a time interval between the two invocations of a method.

→ for Ex:

@Scheduled(fixedRate = 500)

public void doWork()

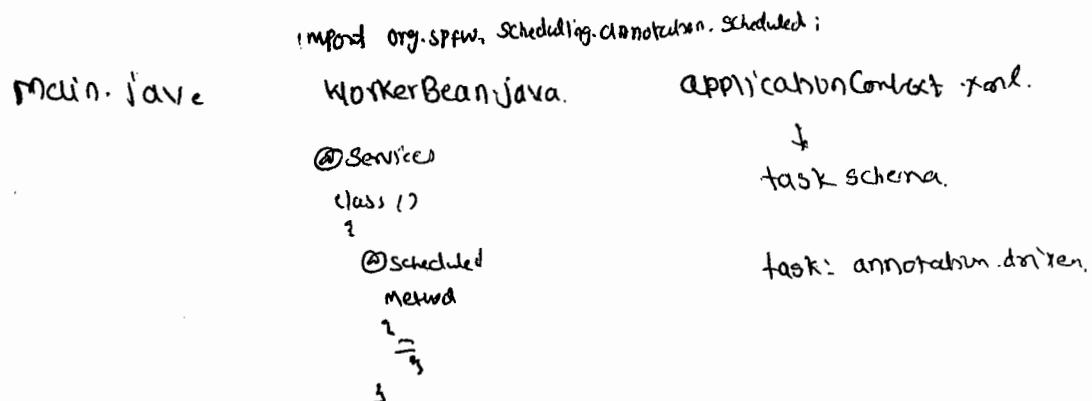
{

=

}

→ doWork() method called by scheduler automatically for every 5 seconds.

→ fixedDelay is a fixed time period between completion of last invocation to the start of the next invocation of the job



cron = " cron expression"

contain 7 parts separated with whitespace.

cron = " \* \* 16 1 \* ? " → 4:0:0: PM

cron = " 0 0 16 1 JAN-APR ? " → 2:0:0: PM

cron = " 0 15,30 14,17 \* \* ? "

0:15:00 PM  
0:30:00 PM  
S:15:00 PM  
5:30:00 PM

cron = " 0 0 18 ? \* FRI "      month  
6:0:0 PM

FRI\_L → Last Friday of ~~the~~ month

FRI #2 → 2nd Friday of ~~the~~ month

cron = " 0 0 0 1 JAN ? 2016 "

0:0:0 AM

## Cron

- to define a schedule for a job efficiently we need to pass cron element to the @Schedule annotation.
- to the cron element we need to pass a cron expression as a value.
- a cron expression contains 7 parts & each part will be separated with white-space.

Cron = "cron expression"

- ① seconds (0-59)
- ② minutes (0-59)
- ③ hours (0-23)
- ④ Day (1-31)
- ⑤ month (1-12) (JAN-DEC)
- ⑥ day of week (1-7 | SUN-SAT)
- ⑦ year (optional)

→ Ex:1 Cron = "0 0 16 1 JAN-APR ?"

4:00 PM

Scheduler will run the job 4PM on 1<sup>st</sup> of ~~the~~ month from JAN to APR.

Ex:2 cron = "0 15,30 14,17 \* \* ?"

2:15:00 PM

2:30:00 PM

5:15:00 PM

5:30:00 PM

A scheduler will run the job at the ~~below~~ <sup>above</sup> times, on every day.

## Main.java

```
? ApplicationContext ctx = new ClassPathXmlApplicationContext ("applicationContext.xml");
}
```

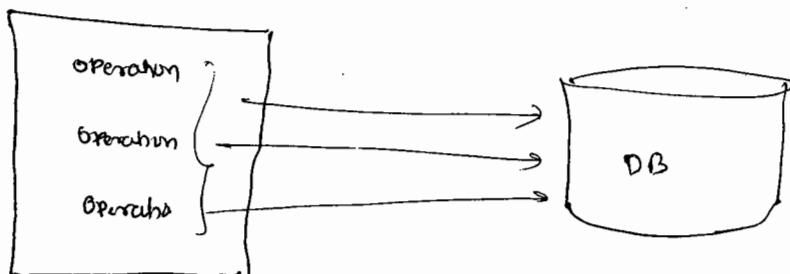
## jar

- ① aopalliance-1.0.jar
- ② commons-logging
- ③ spring-aop
- ④ spring-beans
- ⑤ spring-context
- ⑥ spring-context-support
- ⑦ spring-core
- ⑧ spring-expression
- ⑨ spring-tx

25-4-2014

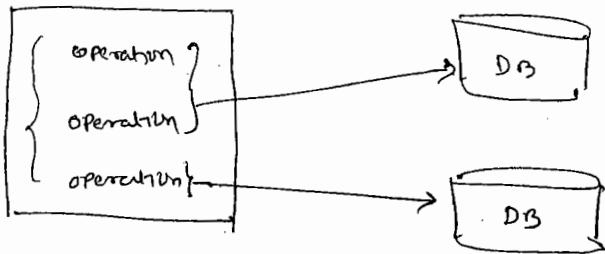
## Transaction Managements

- ① local transaction
- ② global | distributed | xa transaction.



local

one database  
used for  
transaction.



global transaction.

more than one DB used for transaction.

transaction bet<sup>n</sup> same bank is local transaction. One database used.

transaction bet<sup>n</sup> two diff. bank. is global transaction as ~~diff~~ more than one db

- A transaction is a unit of work
- To complete a unit of work it may have one or more operations.
- we can define a transaction as a group of operations that are used to perform a unit of a work.
- Transaction may reach to either a success or a failure point.
- If all operations are executed successfully then a transaction will become a success.
- If any operation is fail then a transaction will become fail.
- If a transaction reached to success then operations are committed on the database. and if it is reach to failure. then the operations are rolled back <sup>on</sup> database.
- In Java we can execute two types of transaction.
  - ① Local transaction.
  - ② global | distributed / XA transaction.
- a local transaction means, a one database will be used for executing the operation of a transaction.

- In a Global transaction, more than one database will be used for executing the operations of the a transaction.
- In Java with JDBC API & the hibernate API we can perform only local transaction.
- With EJB API & Spring, we can perform both local and the global transaction.

Q Why JDBC does not support a Global transaction?

- In JDBC, to manage a transaction, the methods are given in Connection interface.
- To execute a global transaction, in a JDBC Application multiple connections are required.
- If one connection committed, and while committing another connection, if any exception occurs then already committed connection cannot rollback its operations.
- So a Transaction is not going to follow its atomicity property.

Ex:

```

Connection con1 = ...;
Connection con2 = ...;

con1.setAutoCommit(false);
con2.setAutoCommit(false);

Statement stmt1 = con1.createStatement();
Statement stmt2 = con2.createStatement();

try {
 stmt1.executeUpdate("....");
 stmt2.executeUpdate("....");
}

```

```

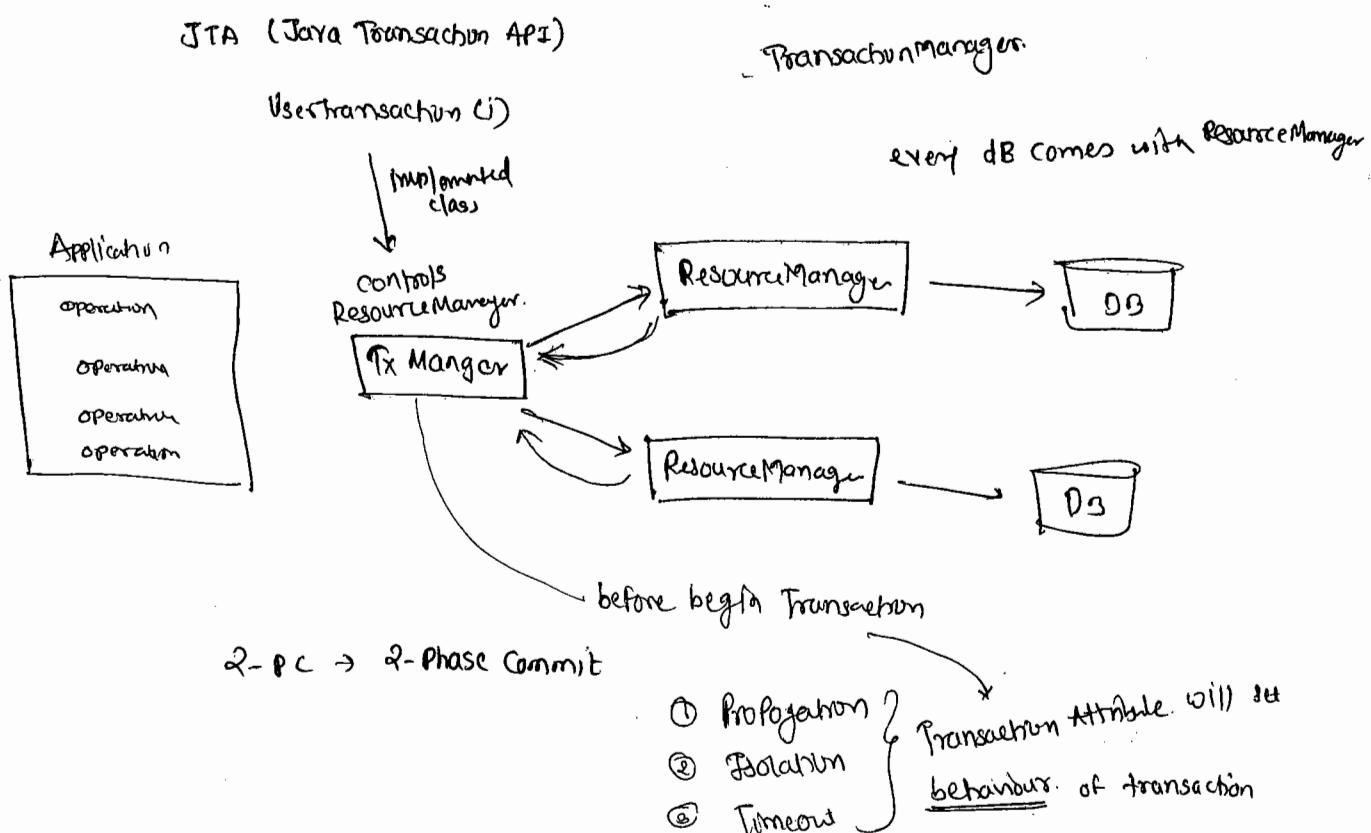
 cont. Commit();
 Con2. Commit();
}

catch (Exception e)
{
 cont. rollback();
 Con2.rollback();
}

```

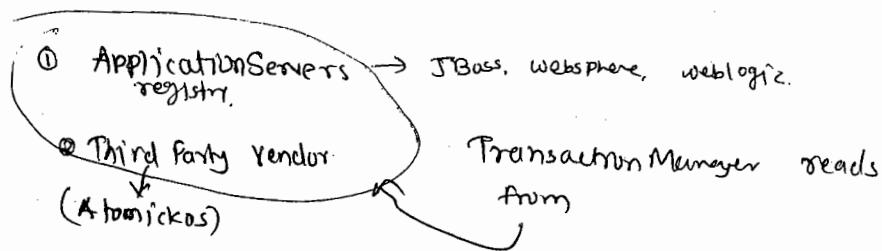
- In the above code if an exception is occurred while committing the Con2, then control enters into catch block but cont cannot be rollback because it is already committed. So with JDBC it is not possible to run global transaction.

## J2EE (JEE)



26-4-2015

## How Global transactions work ?



- To execute Global Transaction, as part of J2EE SUN microsystem has provided ~~JTA~~ JTA (Java transaction API)
- As part of JTA, an interface is given called UserTransaction, an object of its implementation class is called a TransactionManager.
- Implementation for JTA will be provided by Application Servers & Third Party vendors.
- A TransactionManager manages one or more ResourceManagers, to execute a Global Transaction.
- every database will have its own ResourceManager & the TransactionManager controls ResourceManagers.
- the below are the steps that are executed, in a Global Transaction
  - ① an Application read the TransactionManager object or (TransactionManager object) from JNDI registry of application server or an application creates an object of implementation class of UserTransaction.
  - ② When a Transaction begin then TransactionManager tells ResourceManagers to execute the operations temporary on data base.
  - ③ When application asks a TransactionManager to commit then TransactionManager will ask a ResourceManagers whether they are

ready to commit or not.

- ④ if ResourceManager returns "yes" then TransactionManager will issue Commit to the ResourceManager. So that ResourceManager will commit the operations.
- ⑤ If ResourceManager returns "no" then TransactionManager issues Rollback to the ResourceManagers.

### Note

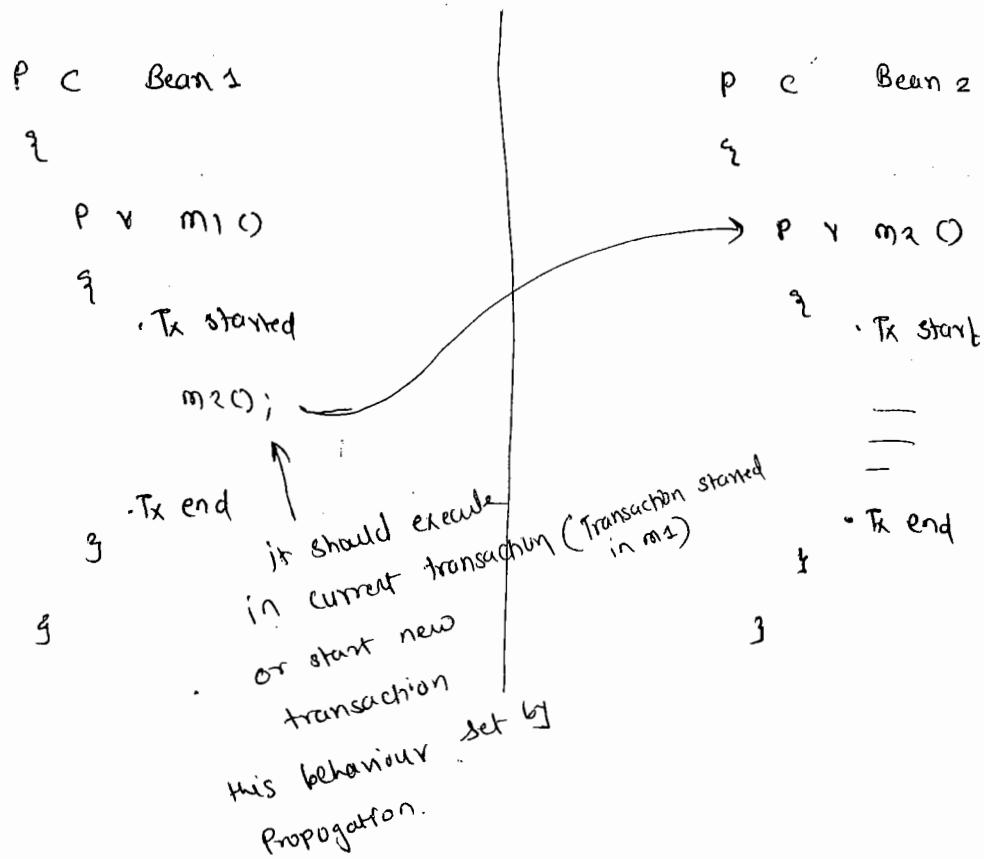
- In Java all application servers comes with a Global Transaction Manager
- When application server started then the server will create TransactionManager object and then it will store the object in registry with key "java:comp/UserTransaction".
- In a Java Application, before a transaction begins the application should read TransactionManager object from JNDI Registry, by calling lookup() method.

### Transaction Attribute

- In a Global Transactions, a Program instructs TransactionManager to begin transaction but a program cannot directly begin a transaction.
- Before a transaction begins we need to define the behaviour of a transaction in the form of Transaction Attributes.

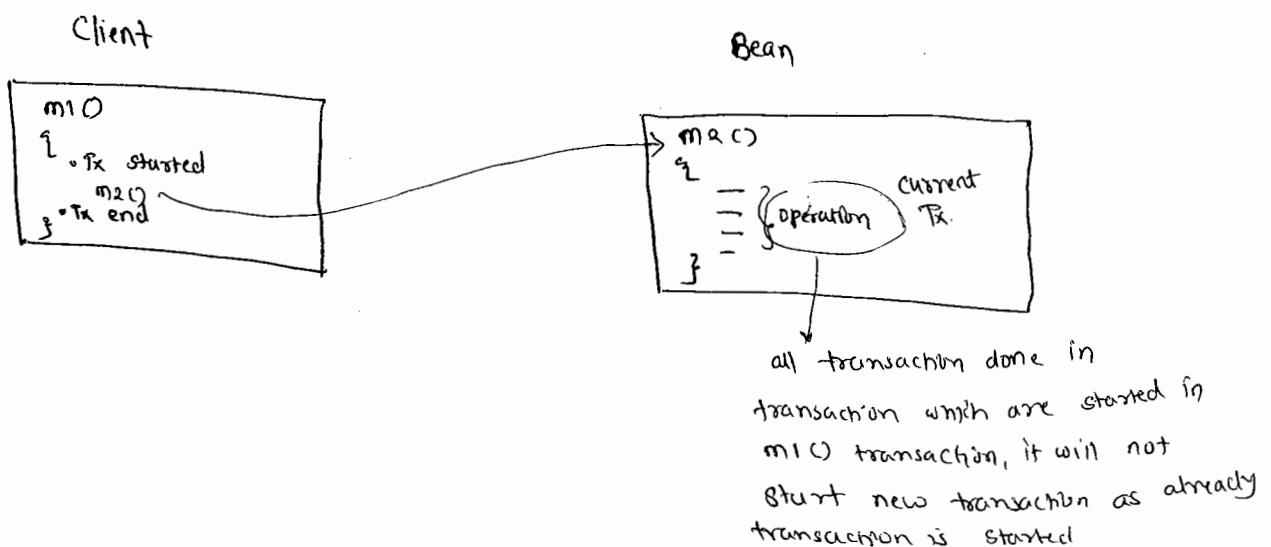
→ Transaction Attributes are.

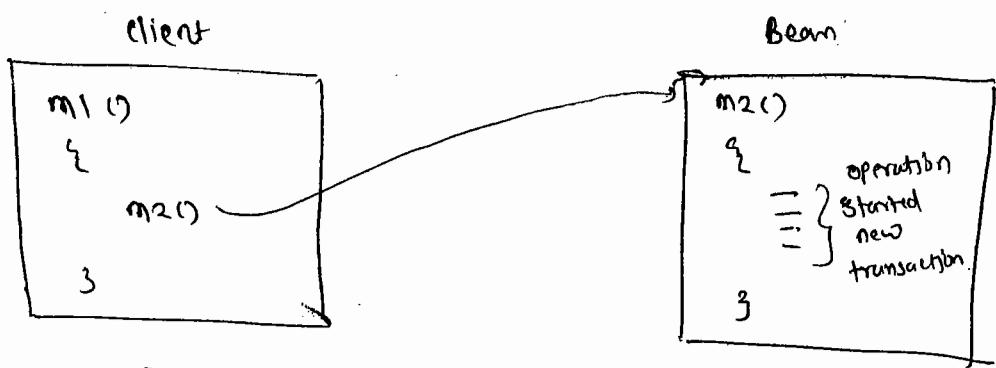
- ① Propagation
- ② Isolation
- ③ Timeout



### Propagation:

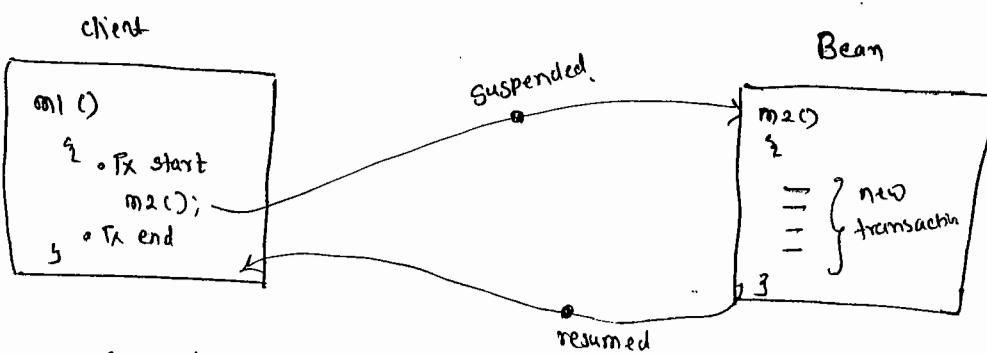
- ① required:



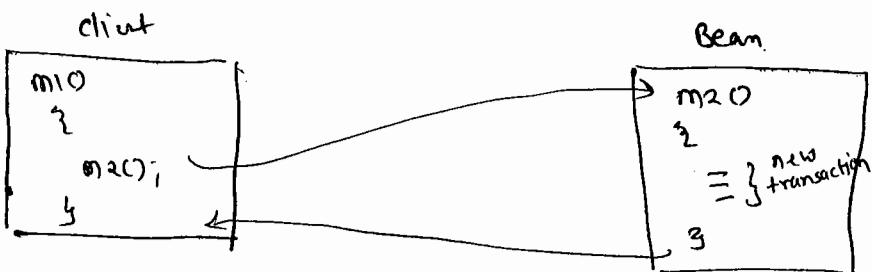


②

Requires new:

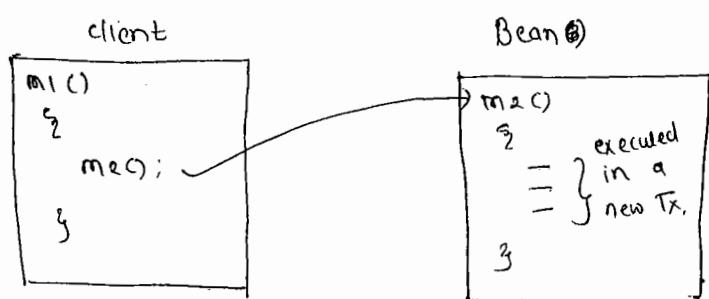
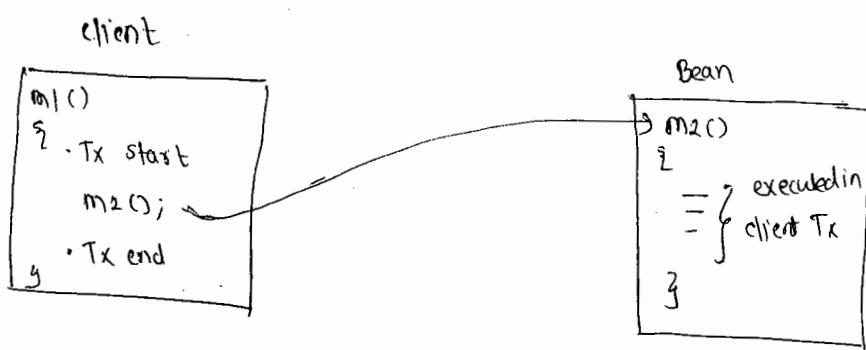


Suspend current transaction  
↳ Started new transaction & after that current transaction resumed.



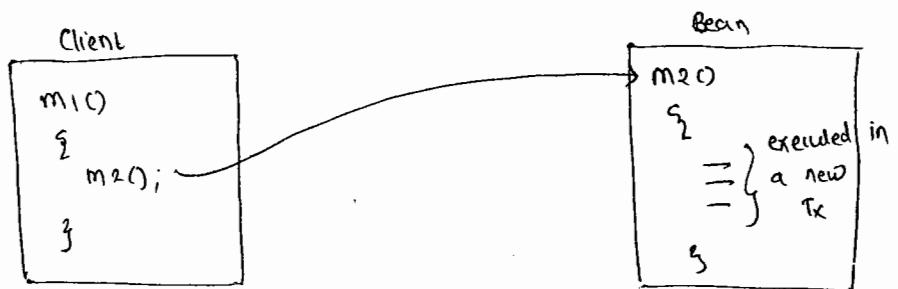
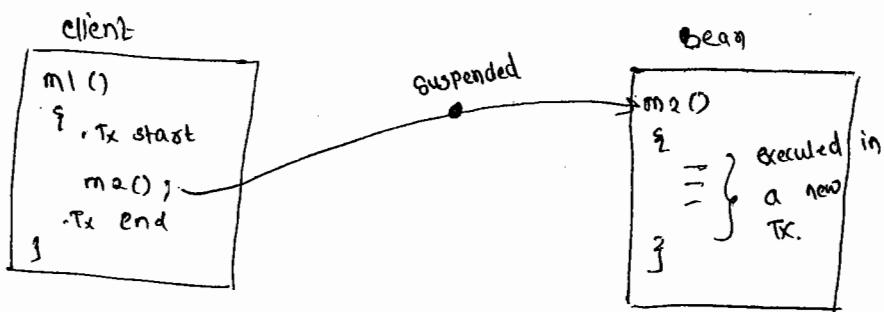
## ① ~~Required~~:

- a TransactionManager finds whether a transaction started at client or not, if started then a transactional method of bean also executed in same transaction.
- If there is no transaction started at client then TransactionManager begin a new transaction for the transaction method.



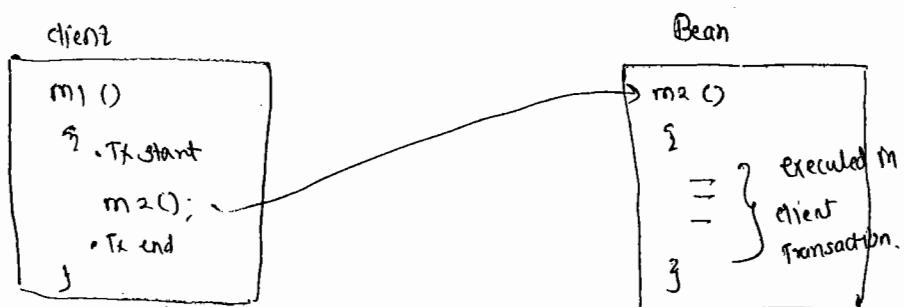
## ② ~~Requires New~~:

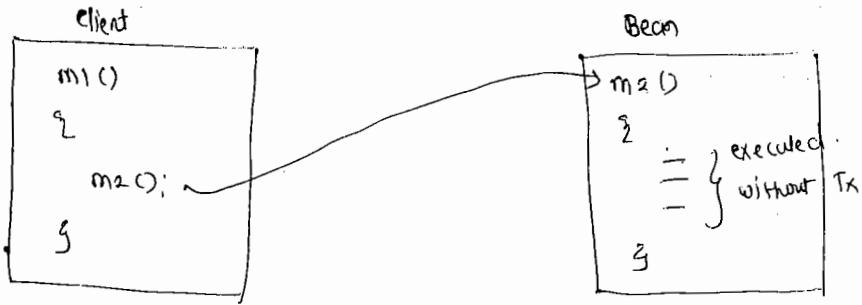
- a transaction manager verifies whether transaction started at client side or not, if started then it suspends it & starts a new transaction for the transactional method of Bean.
- if there is no transaction started in client then a transaction Manager begins a new transaction for the transactional method of Bean.



### ③ supports:

- a transaction manager verifies whether at client side a transaction is started or not, if started then transactional method of Bean also executed on the client transaction.
- if no Transaction started at client then a transactional method of a Bean also executed without a transaction.

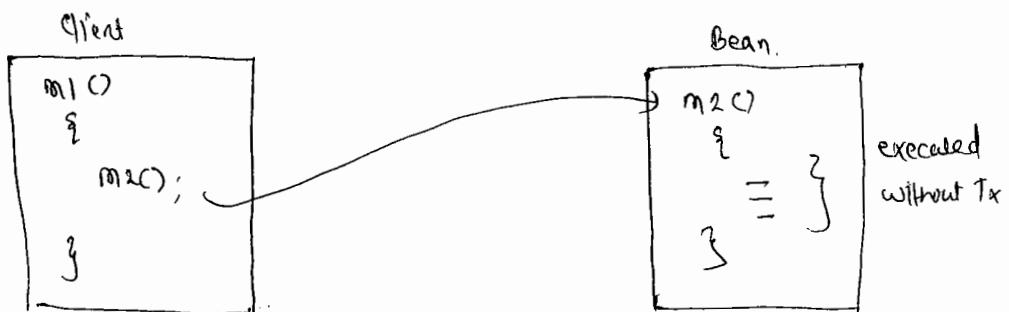
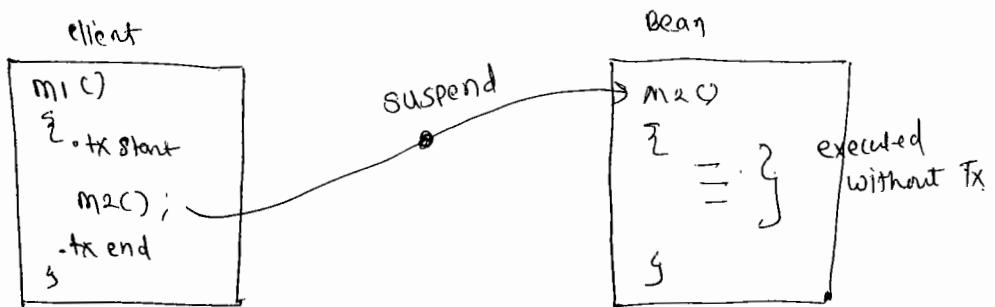




④

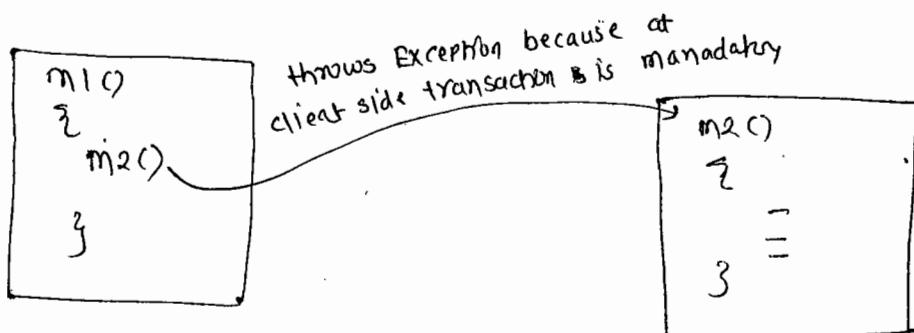
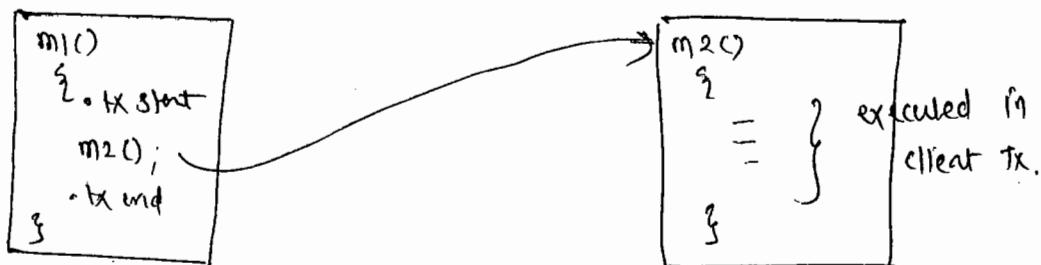
#### not Supported

- a transaction manager verifies whether a transaction started at client side or not, if started then suspends it and executes transactional method of a Bean without a transaction.
- If there is no transaction at client side then a transactional method of a Bean also executed without a transaction.



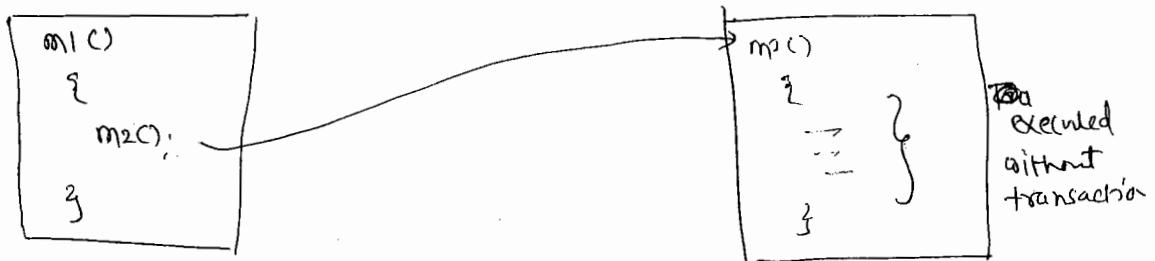
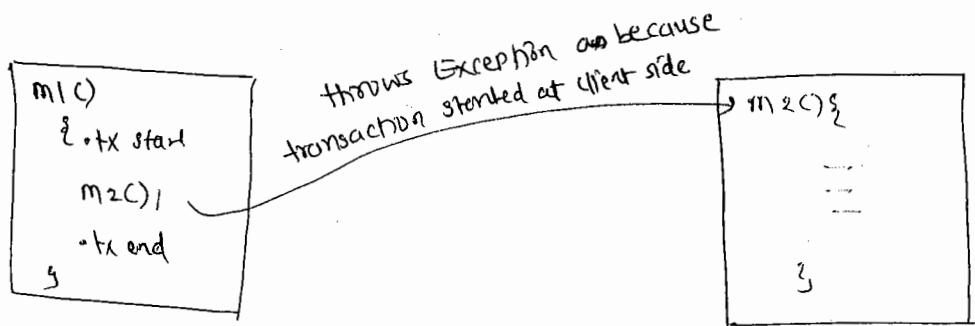
## ⑤ mandatory:

- a transaction manager verifies whether a transaction started at client side or not, if started then executes a transactional method also in current transaction or same transaction.
- If no transaction started at client side then throws an exception, because at client side transaction is mandatory.



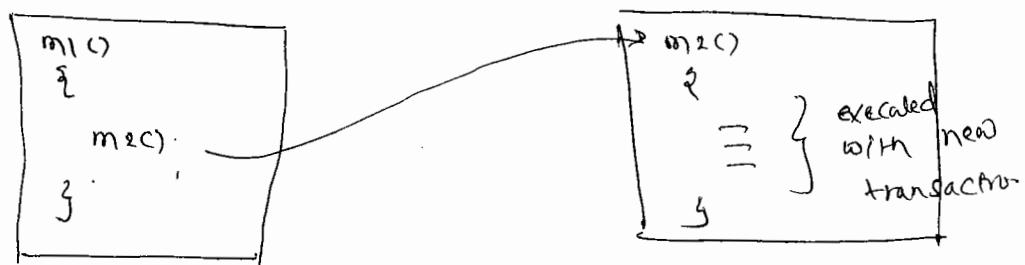
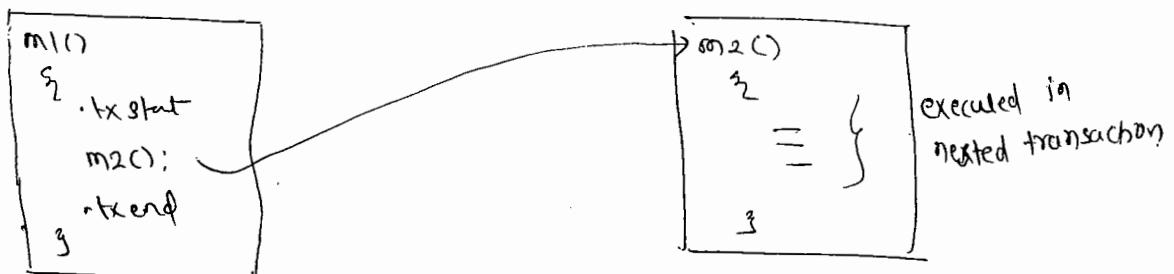
## ⑥ never:

- a transaction manager verifies whether a transaction started at client side or not if started then throws an exception.
- If no transaction started at client side then a transactional method of Bean also executed without a transaction.



## ⑦ nested

- a transaction manager verifies whether a transaction started at client side or not, if started then a transaction manager starts a nested transaction for the transactional method.
- If no transaction started at client, then a new transaction will be started for the transactional method of a bean by transactional manager.



## Concurrency Problems in Transactions

### USERS

uid	uname	age
1	A	30
2	B	28
3	C	25

dirty read  
problem

Tx 1

→ started

—  
—  
—  
—

Select age from USERS  
where uid = 1;  
// age = 32

—  
—

Tx 2

→ started

—  
—

update USERS set age=32  
where uid = 1;

—  
—  
—  
—

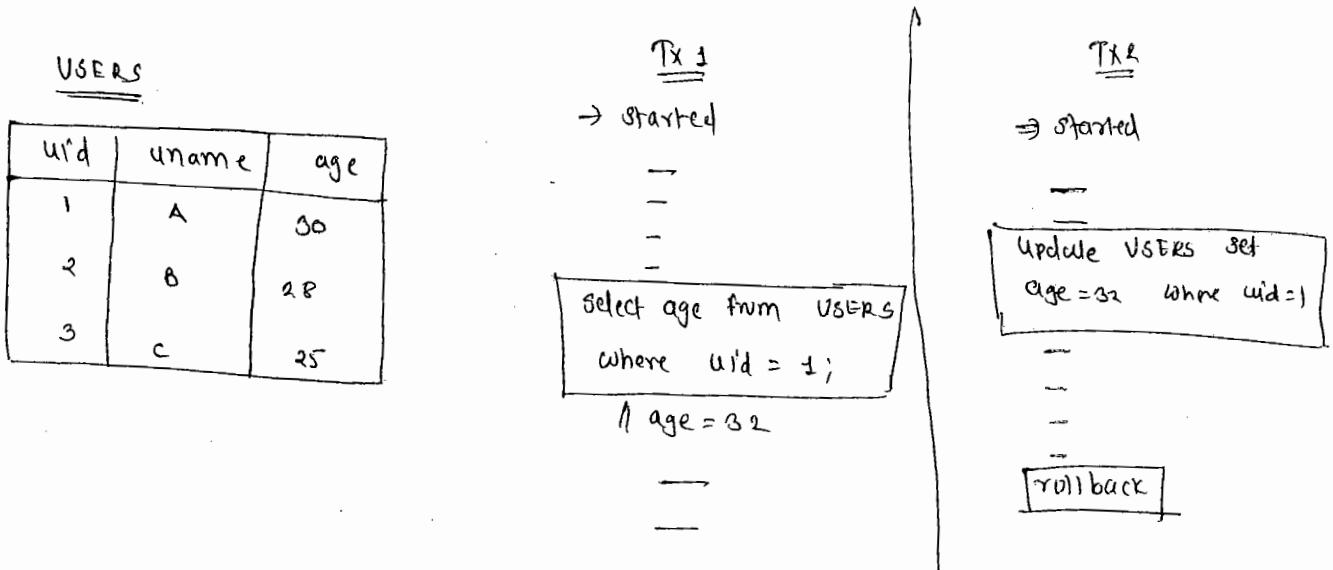
roll back

Concurrency Problems in Transactions

→ by default changes made on data by one transaction will be visible to other transaction when both are concurrently working or executing on the same data.

→ the problems that are going to occur, when multiple transactions are working simultaneously on the same data of the same table are called Concurrency Problem.

### ① Dirty read Problem:



- In the above Tx2 updated age as 32 and after that Tx1 has read.
- later, tx2 has rollback so data read by tx1 is not a valid data. Its a wrong data. this is dirty read Problem.
- dirty read Problem occurs, when transaction is reading a data which is updated by another transaction before the another transaction is committed.
- if another transaction is rollback then the data read by first transaction becomes dirty data.

28-4-2015

② Non-repeatable read

→ This concurrency problem occurs in transactions, when a transaction is reading a same data for multiple times and the if any changes are observed in data from previous read to the current read.

Tx 1

→ Started

Select age from USER where  
uid = 1;

// age = 30

Select age from USERS  
Where uid = 1

// age = 32

Tx 2

→ Started.

—  
—  
—

Update USERS set age = 32  
Where uid = 1;

Commit

—  
—  
—

→ In the above, Tx1 has selected age for twice and the changes are observed in age from 1<sup>st</sup> read to 2<sup>nd</sup> read so it is called a non-repeatable read problem.

③

## Phantom Read Problem.

- This Concurrency Problem occurs when a transaction is reading a data within a range for two ~~at~~ times. and if any extra rows are identified at the 2<sup>nd</sup> time than 1<sup>st</sup> time.
- the extra rows identified are called Phantom rows.

Tx1

→ Started

```
Select * from USERS where
age between 20 and 28;
```

// 2 rows

—  
—  
—

```
Select * from USERS where
age between 20 and 28;
```

// 3 rows

Tx2

→ Started

—  
—  
—

```
insert into USERS values
(4, 'D', 26);
```

Commit

—  
—  
—

→ In the above, tx1 is selecting the rows for a times bet<sup>n</sup> age 20 & 28, and the 2<sup>nd</sup> time 1 extra row is identified this is happen because another concurrent transaction tx2 inserted a row matching the condition.

→ Here the Problem is called a Phantom read Problem.

## Isolation level

Isolation level	dirty read	non-repeatable read	phantom read
read uncommitted	✓	✓	✓
read committed	✗	✓	✓
repeatable read	✗	✗	✓
Serializable	✗	✗	✗

- transaction concurrency problem are resolved by setting an Isolation level to a transaction.
- In Java there are 4 isolation level defined
  - ① read uncommitted (default)
  - ② read committed
  - ③ repeatable read
  - ④ Serializable
- read uncommitted Isolation level allows all the 3 concurrency problem in a transaction.
- read committed prevents dirty read problem; but allows non-repeatable problem & phantom read problem.
- repeatable reads ~~problem~~ prevents ~~not~~ dirty read & non-repeatable read, but allows phantom read problem.

- In case of repeatable read a transaction will lock the row while selecting the data, so other transaction can not update or delete that row.
- Serializable Prevents all concurrency problem.
- In case of Serializable, Read a transaction will lock the table while selecting the data, so other transactions can not insert any rows to the table, until the lock is released.

## \* Spring Transaction Management

- In spring fw a transaction manager is required for both local and global transaction.
  - the framework has given multiple predefined transaction manager classes & we need to configure a suitable transaction manager class in a configuration file based on the persistence technology used in the transaction.
- ① Jdbc → DataSourceTransactionManager
  - ② hibernate → HibernateTransactionManager
  - ③ Jdo → JdoTransactionManager
  - ④ Jta → JtaTransactionManager.
- In Spring framework a transaction service can be add to a method in 2 ways.
    - ① declaratively (using xml configuration)

## ② using Annotations.

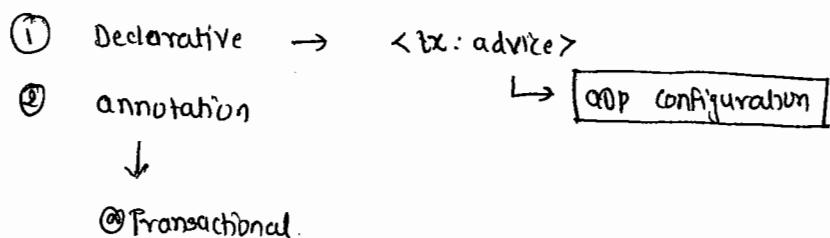
### Declarative Approach

- Spring framework has provided a special advice called transaction advice to configure the transaction attributes.
- AOP configuration is used for adding transaction advice to a methods of Bean, by identifying the method through a pointcut

29-4-2015

### Spring transaction.

① Local      } Transaction Manager.  
② Global-



- In Declarative Approach a transaction advice can be added to business method through AOP configuration.

→ Next Page

```

<tx: advice id = "advice1" transaction-manager = "txm">
 <tx: attributes>
 <tx: method name = "method1">
 propagation = "REQUIRES_NEW"
 isolation = "READ_COMMITTED"
 timeout = "180"
 </tx: method>
 </tx: attributes>
</tx: advice>
<app: config>
 <aop: pointcut id = "pointcut1">
 expression = "execution (* com.saty.spring.transaction.*.*(..))" />
 <aop: advisor advice-ref = "advice1">
 pointcut-ref = "pointcut1" />
 </aop: advisor>
</app: config>
<beans>

```

## Annotation Approach

- In order to reduce the XML configuration in Spring Configuration file, the framework has given **① Transactional annotation**
- **① Transactional** is a method level annotation.
- this annotation has 3 elements to define the Transaction attribute

- |                                           |                                           |
|-------------------------------------------|-------------------------------------------|
| ① Propagation<br>② Isolation<br>③ Timeout | ① Propagation<br>② Isolation<br>③ Timeout |
|-------------------------------------------|-------------------------------------------|

- Propagation value defined in an Enum called Propagation.
- Isolation value defined in an enum called Isolation.
- ~~Timeout value defined in a connection interface~~

② Transactional (Propagation = Propagation.REQUIRED,  
 Isolation = Isolation.SERIALIZABLE,  
 Timeout = 5)

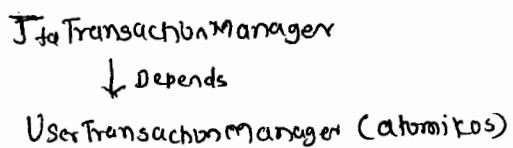
Public void method1()

{

—  
—

}

30-4-2015



DataSource ds = new xxx();

ds.setXxx();

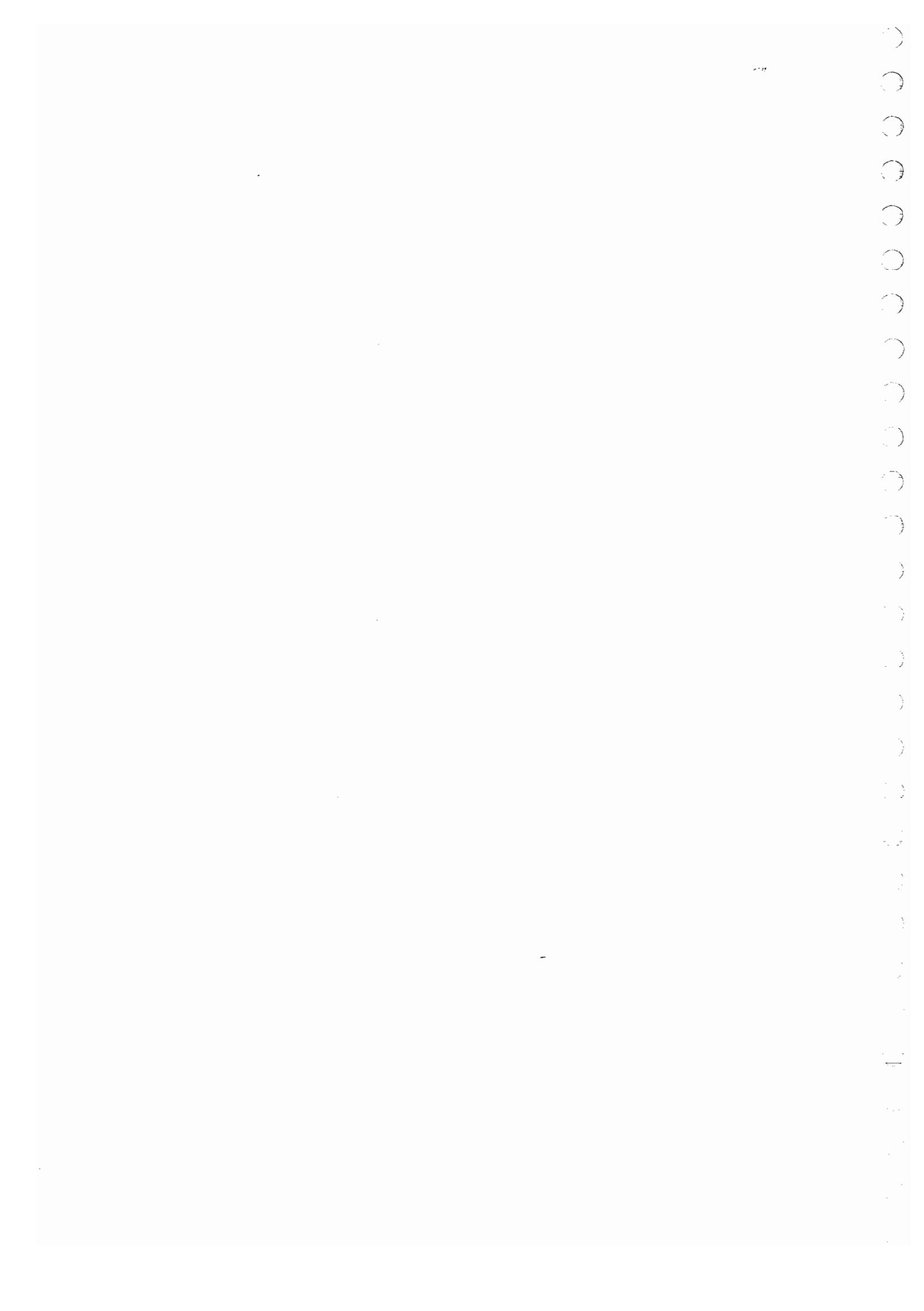
Connection con = ds.getConnection();

ConnectionPoolDataSource cpds = new xxx();  
 cpds.setXxx();

PooledConnection pc = cpds.getPooledConnection();

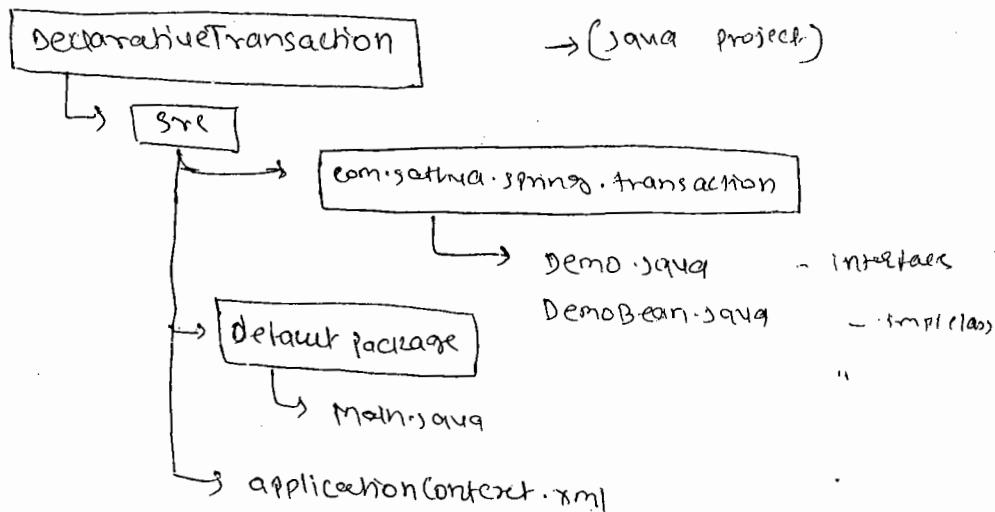
## Global Transaction in Spring

- a global transaction means multiple resources are involved in a transaction.
- In spring fw, ~~Jta~~ JtaTransactionManager class is given to manage the distributed transaction.
- JtaTransactionManager depends on a global transaction manager provided by app server or a third party to manage a global transaction.
- Atomikos is a 3rd party which provided implementation for ~~Jta~~ Jta.
- Atomikos has given UserTransactionManager, it is an implementation class of UserTransaction Interface as a global transaction manager. so if Atomikos API is used then JtaTransaction Manager of spring depends on UserTransactionManager of Atomikos to manage a global transaction.
- a Connection ie open by using an implementation class of XADataSource Interface can only participate in global transaction.
- Atomikos API has given ~~Atomikos~~ AtomikosDataSourceBean, which uses implementation class of XADataSource, creates a connection with database & stores them in a ~~con~~ connection pool.
- When developing global transaction app in spring, JdbcTemplate of spring reads a connection from a connection pool created by AtomikosDataSourceBean.



\* Directory structures for handout of Spring-service [Local transaction] 11/5/15

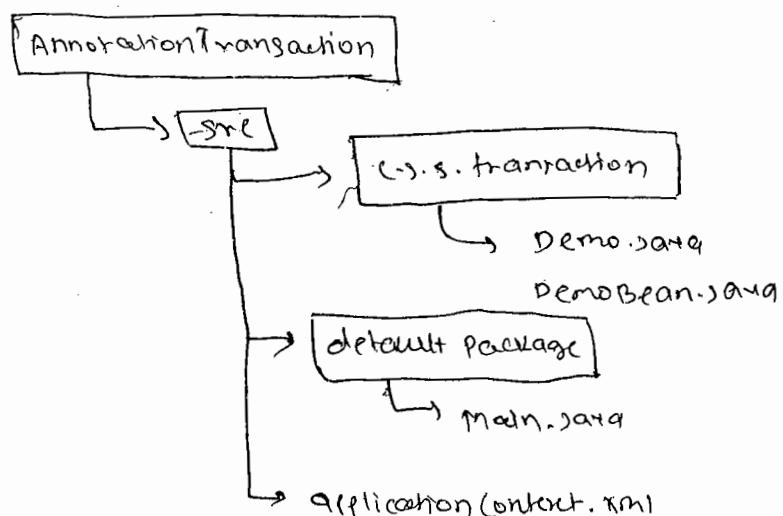
## ① Declarative Transaction [Page 2].



\* jar's list

- ① ojdbc14.jar
- ② spring-aop-3.2.4
- ③ spring-beans-3.2.4
- ④ spring-context
- ⑤ spring-context-support
- ⑥ spring-core
- ⑦ spring-expression-3.2.4
- ⑧ spring-jdbc-3.2.4
- ⑨ spring-tx-3.2.4
- ⑩ aspectjweaver-1.6.2
- ⑪ aspectjrt-1.6.9
- ⑫ commons-logging-1.0.4
- ⑬ aopalliance-1.0.jar

## ② Annotation Transaction



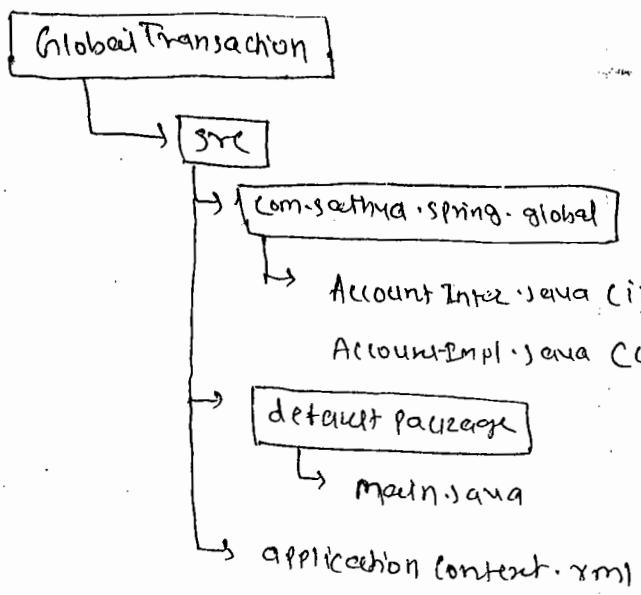
// prefer spring-services handout. ②.

\* follow the same jar of about ex ①.

## (11) Global Transaction application

Refer Page 112/113

11/5/15



JAR's (buildpath)

- ① commons-logging-1.0.4.jar
- ② spring-aop-3.2.4.jar
- ③ aspectjrt-1.6.9.jar
- ④ aspectjweaver-1.6.2.jar
- ⑤ aspectjrt.jar
- ⑥ transaction-jta-3.7.0.jar
- ⑦ automates-transaction-api.jar
- ⑧ automates-util.jar
- ⑨ transaction-3.7.0.jar
- ⑩ jta.jar
- ⑪ spring-tx-3.2.4.jar
- ⑫ s-bean.jar
- ⑬ s-context.jar
- ⑭ s-c-support.jar
- ⑮ spring-core.jar
- ⑯ spring-expression-3.2.4.jar

⑰ mysql-connector-java-5.1.0-bin.jar

- ⑲ spring-jdbc-3.2.4.jar
- ⑳ aspectjweaver-1.6.9.jar

Note: Before executing global transaction application we need to create account1 table in oracle and account2 table in mysql as ready.

### ① In Oracle DB

```

SQL> create table Account1 (accno number(5), primary key, bal number(5));
SQL> insert into Account1 values (101, 5000);
SQL> commit;

```

### ② In MySQL DB

```

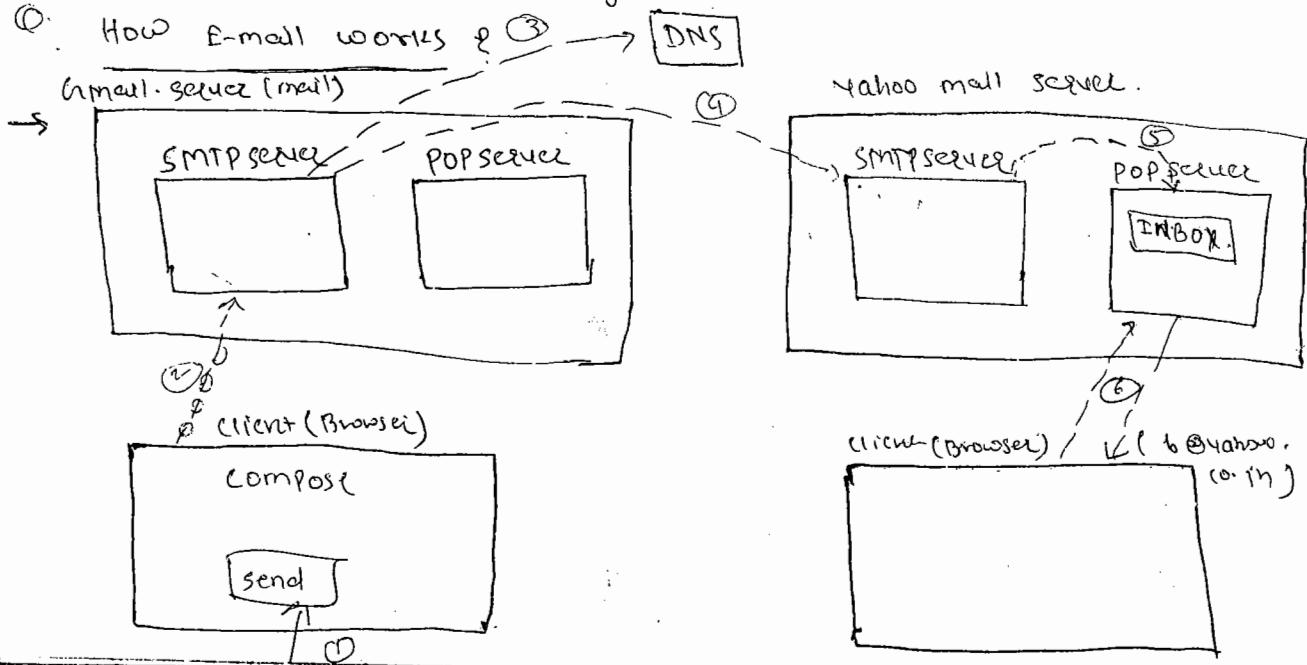
SQL> create table Account2 (accno int, primary key, bal int);
SQL> insert into Account2 values (102, 1000);
SQL> commit;

```

## \* Spring mail :-

11/5/15

- As a part of J2EE, sun-microsystem provided a service api called java mail api for sending an e-mail from a java application and for reading a mail from a java application.
- Java mail api consists a set of interfaces and classes and they are given under two packages ..
- ① javax.mail
  - ② javax.mail.internet
- Using java mail api, to send an e-mail or to read a mail, there is a need to write boilerplate code.
- Spring provided an abstraction layer to avoid the boilerplate code and given spring mail api.
- Spring mail api is only given for sending an e-mail from a spring application, but not for reading an e-mail.
- When developing java projects, there is a need to send the mail to a customer mail account after processing the request. But there is no need for an api to read a mail. Because a customer can read the mail directly through browser.



- ① A user compose the mail and then click as on send button.
- ② A mail will be created as a message object and then it will be send to SMTP server (outgoing).
- ③ SMTP server checks whether receiver or mail account belongs to same mail server or another. If another then finds the location through DNS. (Domain Name server).
- ④ SMTP server will send the message to SMTP server of other mail server.

⑤ SMTP server will send the message to POP server (incoming server). POP server stores message like a file in INBOX folder.

⑥ A receiver connects with POP server and reads a mail from inbox.

21/5/15

→ In Spring mail api, a class JavaMailSenderImpl is given to send an email. So, our Spring bean class depends on JavaMailSenderImpl class.

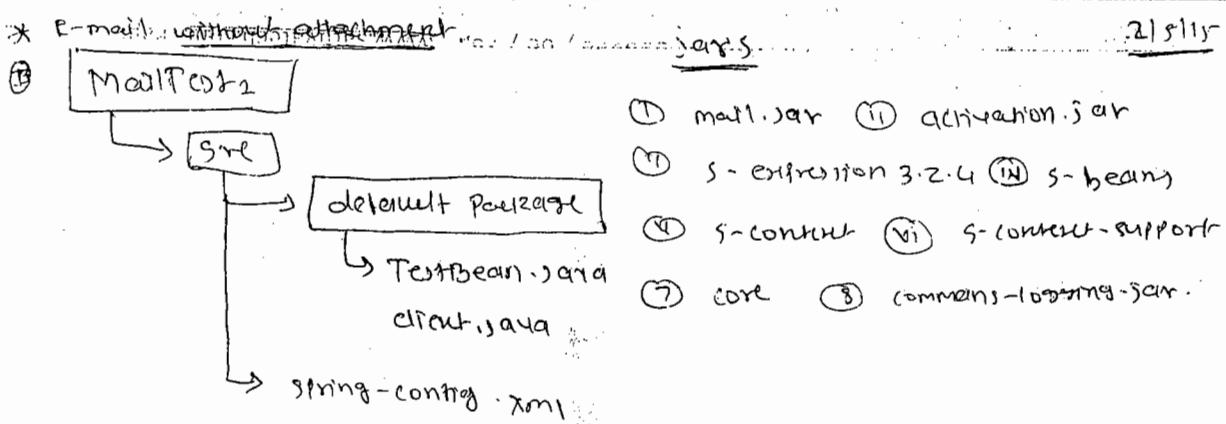
→ JavaMailSenderImpl class has 4 properties -

① Host ② Username ③ Password ④ JavaMailProperties.

→ In spring configuration file we need to configure JavaMailSenderImpl class within its properties.

→ In Spring framework, SimpleMailMessage class is given to create a message without attachments.

→ We need to create an object of SimpleMailMessage & then we need to set the properties like - from, to, subject and text, to prepare a message, sending it is an email.



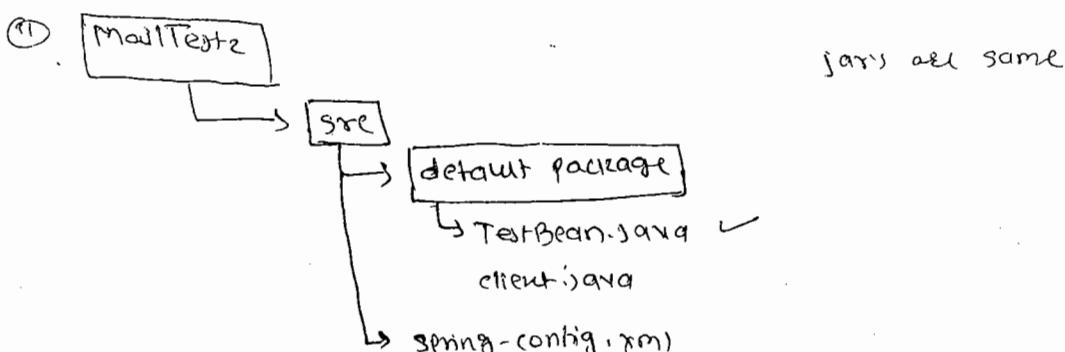
→ Prefer page ⑤ of spring-service handout.

### \* Mail with Attachment :-

To send an email with text and with attachment, we need an object of MimeMessage class.

→ MimeMessage class is from Java Mail API.

→ To set the text and also to add attachment to the MimeMessage class object, we need to use MimeMessageHelper class given by Spring framework.



#### ① TestBean.java

```

public class TestBean {
 private JavaMailSenderImpl mailSender;
 public void setMailSender (JavaMailSenderImpl mailSender) {
 this.mailSender = mailSender;
 }
 public void sendEmail() {
 try {

```

```

● MimeMessage msg = mailSender.createMimeMessage();
● MimeMessageHelper helper = new MimeMessageHelper(msg, true);
● helper.setFrom("boraveninhan@gmail.com");
● helper.setTo("boraveninhan43@gmail.com"); // or yahoo or anything...
● helper.setSubject("mail from spring application");
● helper.setText("This is a test mail with attachment");
● FileSystemResource tsr = new FileSystemResource("location of file");
● helper.addAttachment("demo.java", tsr); // D:\hibernate\Insert.java
● ("Insert.java", tsr);
● mailSender.send(msg);
● } System.out.println("mail successfully sent");

```

catch (Exception e)

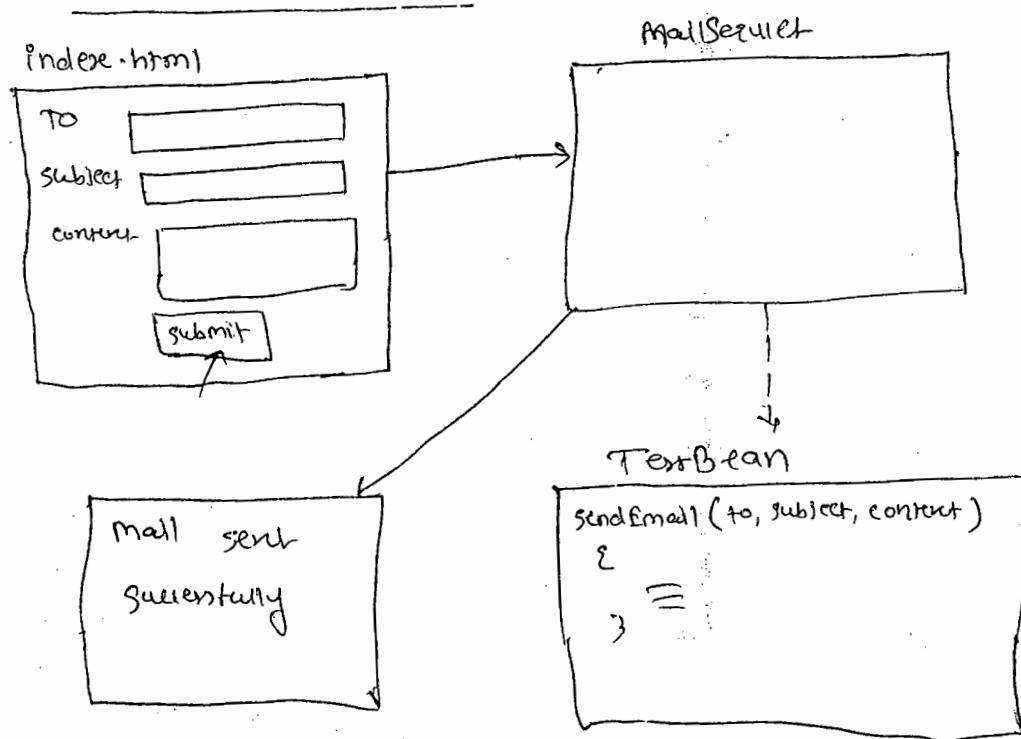
{ System.out.println(e); }

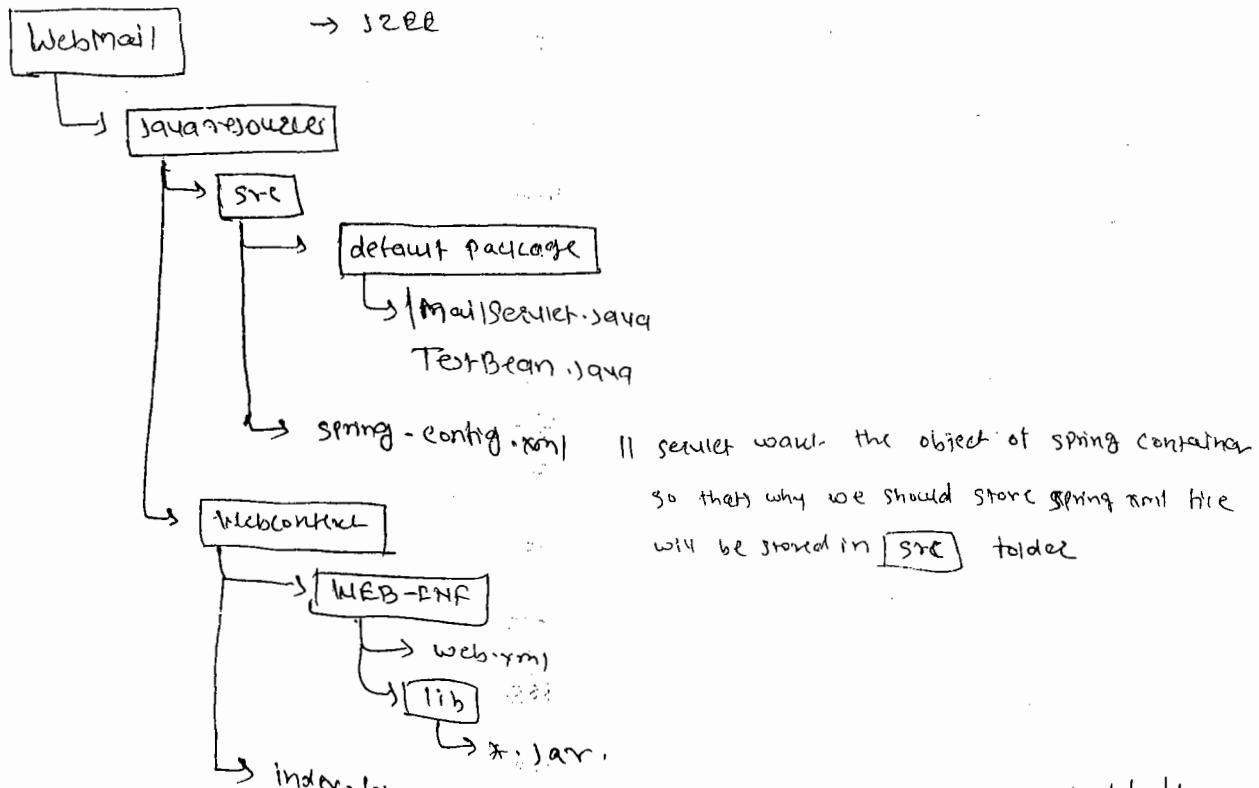
}

→ copy naming file from previous example of mail.

→ If you want to add more attachment then use more filesystemResource

### III) Spring-seculet integration [J2EE]





4/5/15

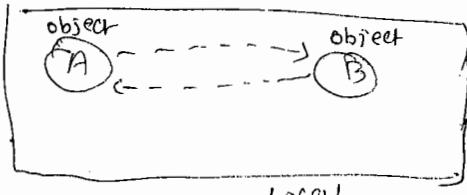
①

class A

Spring - Remoting

(class A & class B are working on  
same machine or JVM, Local)

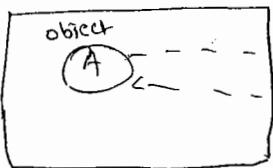
JVM {Local communication on JVM}



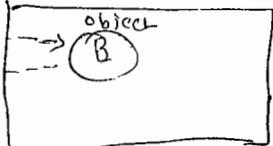
local

②

JVM-1



JVM-2



Remote communication

③

Spring - RMI

④ Spring = HttpInvoker

Stub object = proxy object - [mediator who can't see the service and to  
implement this stub registry required]

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○

○



4-5-2015

## \* Spring-Remoting Service \*

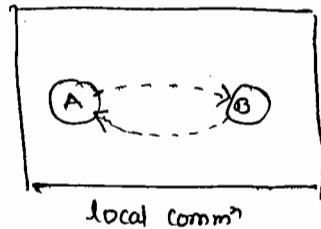
Class A

```

B b = new B();
void m1()
{
 b.m2();
}

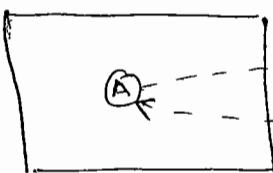
```

JVM

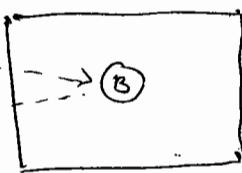


3

JVM-1



JVM-2



Remote Comm

**TIPS**

- Spring-EMI
- Spring-HTRInvoker

- In a Java Application, if two objects are communicating with each other within a same JVM, then we call it as local comm?
- If two objects running on 2 different JVM's are communicating with each other then it is called as remote comm?

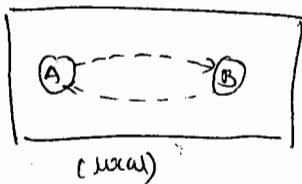
Class A

```

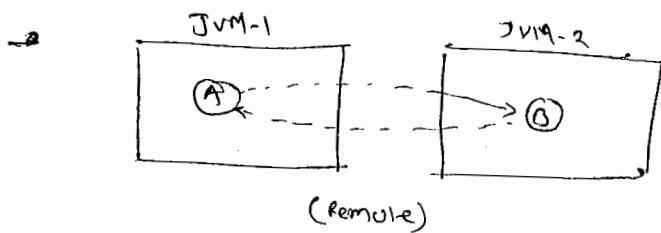
B b = new B();
void m1()
{
 b.m2();
}

```

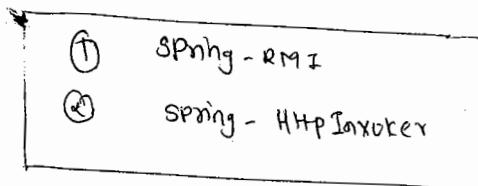
JVM



3

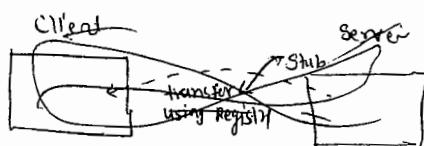


→ In spring framework, we use below two techniques to get remote comm' bet'n to two spring beans.



### Spring-RMI

→ RMI is a distributed technology given as part of Java SE for communicating & objects that are running in two different JVM.



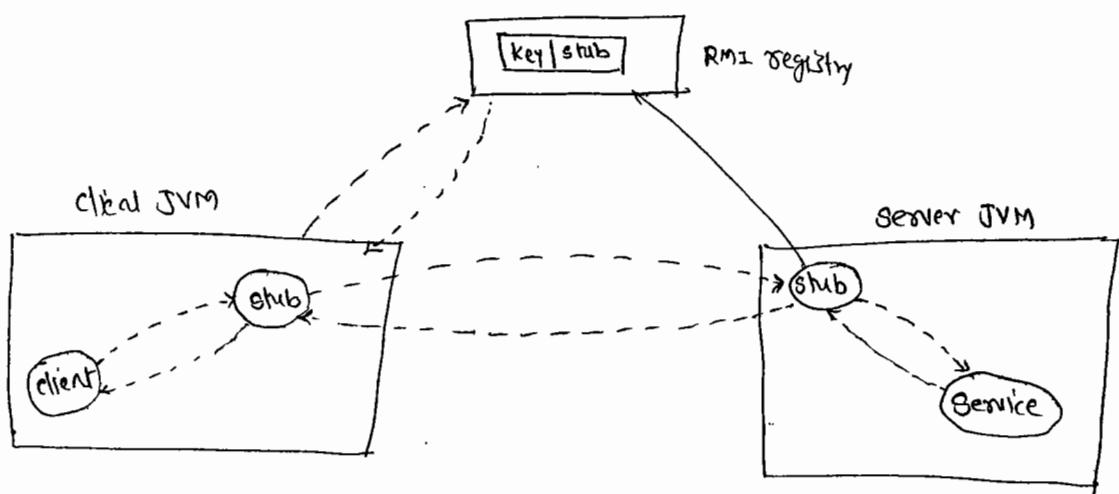
→ In Remoting, a client object can not directly call service, because Service object is running on another JVM.

→ To provide the communication bet'n client & service object in the middle Helper objects are required & they are called Stubs.

→ a stub object will be created at server & then it will be stored in RMI Registry <sup>with</sup> some key.

→ at client side, the application reads the stub object from registry

- the stub object at client & server will do the marshalling & unmarshalling.
- marshalling is a process of converting character data to binary & unmarshalling is a process of converting binary data to character data.
- a typical RMI communication diagram will be like the following.



client: (Stub)

① char  $\xrightarrow{\text{to}}$  Binary  
(marshalling)

request

server: (Stub)

① binary  $\rightarrow$  char  
(unmarshalling)

② binary  $\rightarrow$  char  
(unmarshalling)

response.

② char  $\rightarrow$  binary  
(marshalling)

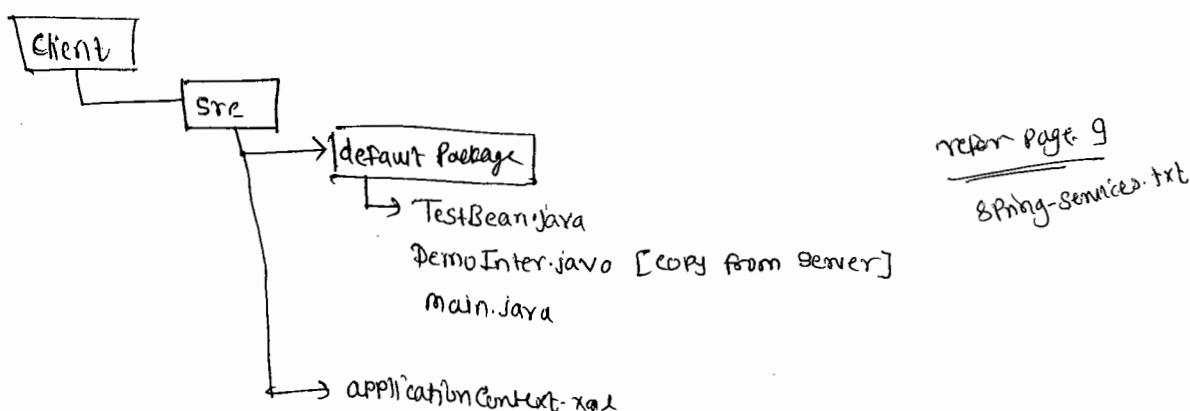
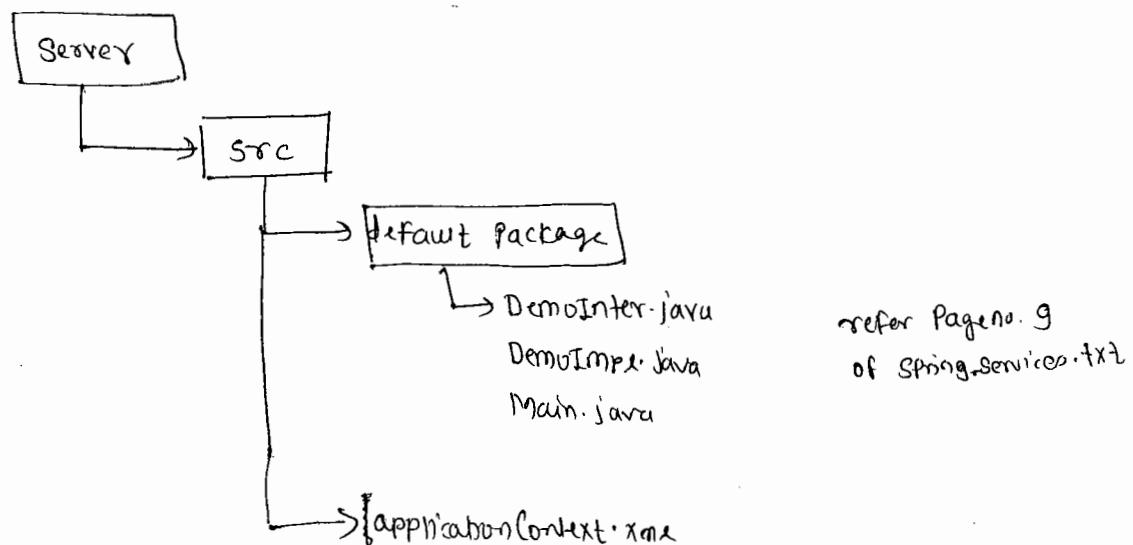
→ Spring-RMI is an abstraction layer on top of RMI technology.

→ If we directly work with RMI technology to create a service we need to follow the below rules.

→ ① we need to extend the Service interface from `java.rmi.Remote` (It is a marker interface).

- ② we need to extend Service class from a Super class `UnicastRemoteObject`.
  - ③ we need to generate a Stub using RMI compiler (~~javac~~ rmic)
  - ④ we need to bind the stub object with RMI registry.
  - ⑤ we need to start RMI registry.
- In Spring-RMI layer, ~~RMI~~ `RmiServiceExporter` class is given and it will do all the above steps automatically so there will be no burden on a server side programmer.
- In a Spring configuration file, we need to configure ~~RMI~~ `RmiServiceExporter` with required properties.

5-5-15



Jar (add in both client & Server)

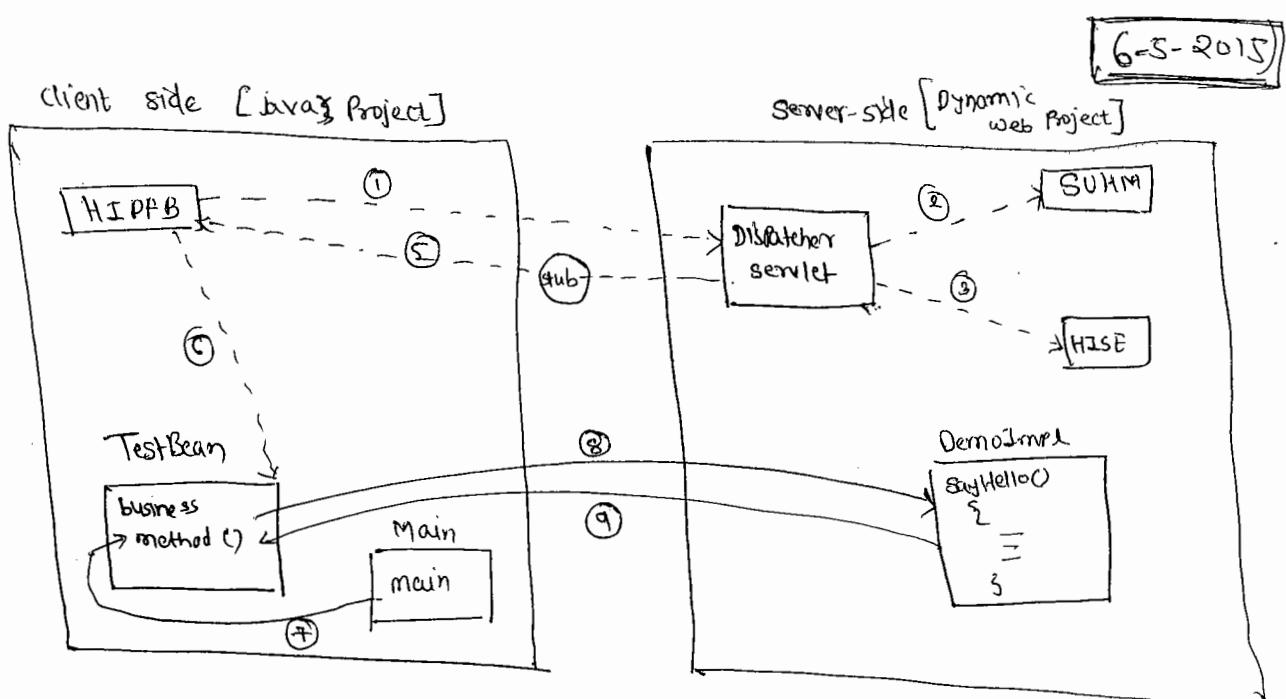
- ① aspectj
- ② commons-logging
- ③ spring-expression
- ④ spring-aop
- ⑤ spring-beans
- ⑥ spring-context
- ⑦ spring-context-support
- ⑧ spring-core

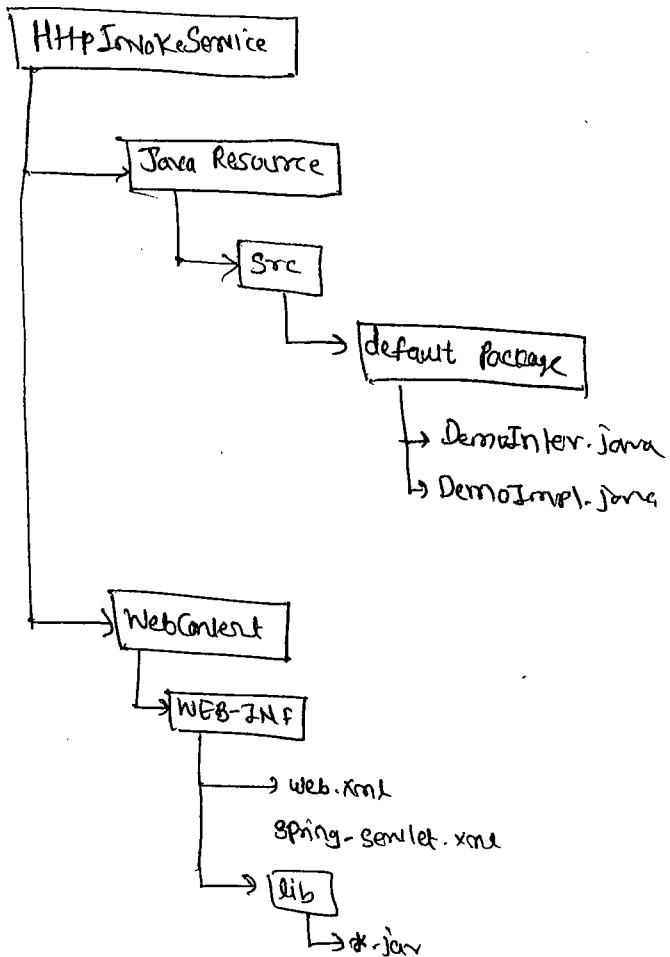
## Spring HttpInvoker:

- In Spring-Rmi Rmi protocol is used for connecting with RMI registry and to read the stub object from RMI Registry.
- RMI protocol is not a firewall-friendly it means if firewall is enable in system in which RMI Registry is running then RMI protocol can not connect to registry so connectivity fails.
- To overcome the above problem framework has given its own remoting technology as Spring-HttpInvoker.
- In Spring-HttpInvoker, HTTP protocol is used to read a stub object from Server. HTTP is a firewall friendly protocol so the comm' is possible always.
- To make a Server side bean accessible to all the client application through http protocol, we need to deploy server side bean in a Web or application server.
- To deploy a server side bean in webapp server, we need to store serverside bean in a webapp & then we need to deploy

that server side bean in a Server.

- In this Spring Http-Invoker, DispatcherServlet of spring MVC is used to receive a request coming from client application.
- To create a Stub class and to return a stub object for a service bean, spring framework has given a predefined class `HttpInvokerServiceExporter`.
- When a request comes from client application then the below steps are executed:
  - ① DispatcherServlet stops the request.
  - ② DispatcherServlet calls HandlerMapping Bean.
  - ③ HandlerMapping bean finds appropriate bean to handle.
  - ④ DispatcherServlet calls the ~~bean~~ `HttpInvokerServiceExporter` bean.
  - ⑤ `HttpInvokerServiceExporter` returns a Stub object & the Stub object will return to client Application by DispatcherServlet.
- To call a DispatcherServlet & to read Stub Object, a client application uses a predefined bean class `HttpInvokerProxyFactoryBean` bean.

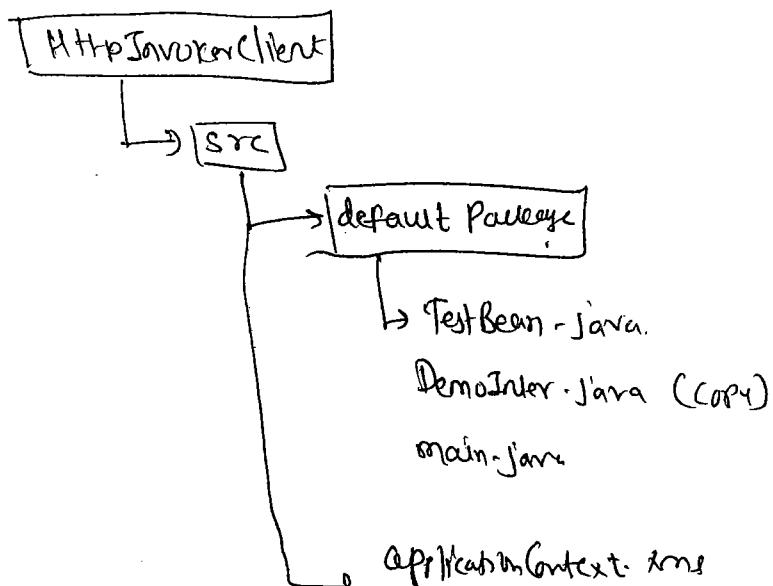




bots

- ① aopalliance
- ② Spring-aop
- ③ commons-logging
- ④ spring-context
- ⑤ spring-context-support
- ⑥ spring-beans
- ⑦ spring-core
- ⑧ spring-expression
- ⑨ spring-web=4.0.5
- ⑩ spring-web-mvc

## ② Client application in Java



Refer Page 11



## JMS Terminology

7-7-2015

- ① JMS provider:- JMS provider is a vendor who provides implementation for interfaces. The implementation given by vendor is called a JMS server  
→ all Java application server are JMS providers.

→

- ② JMS Client:

→ It is a Java application used for either sending a message or for receiving a message

- ③ JMS Administered Objects:

→ A ConnectionFactory and the Destination object are called Administered Objects of JMS.

→ A JMS provider will provide a tool to configure Administered Objects of JMS.

→ An Administered Object of JMS configure Administered Objects & stores them in JNDI Registry.

→ In JMS there are two types of Destination object

- ① Queue ② Topic.

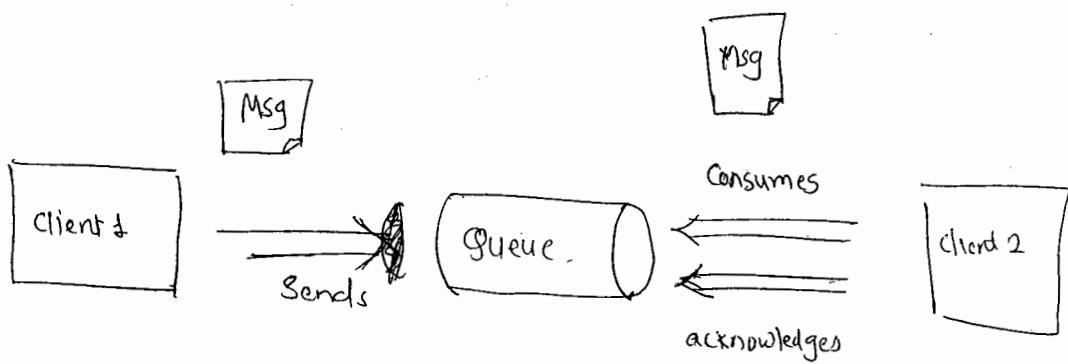
→ Queue is a destination object to store the object in Point to Point model.

→ Topic is a destination object to store the messages in Publish Subscribe model.

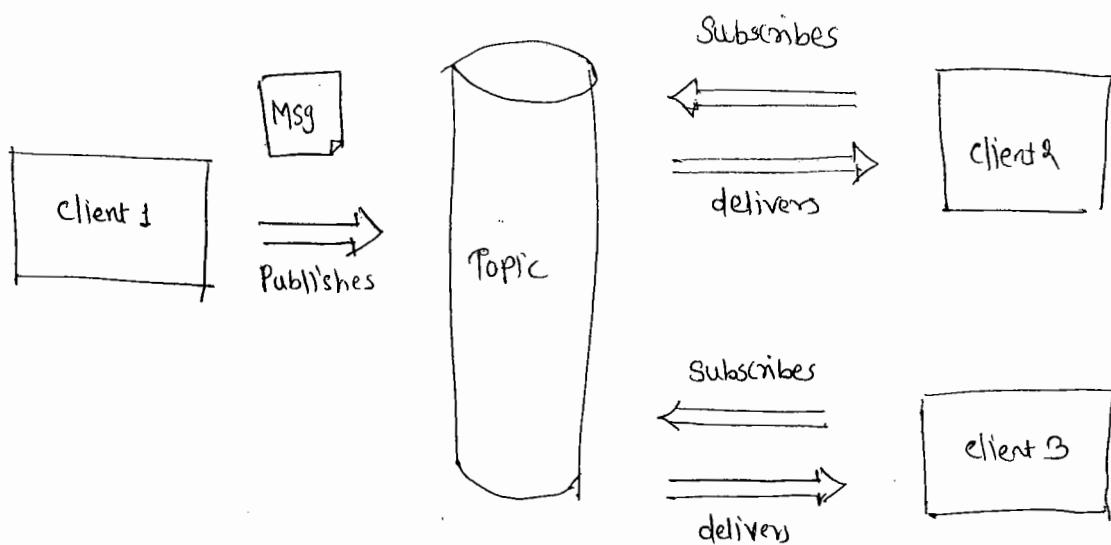
→ JMS support 2 types of messaging model

- ① Point to Point or one to one.  
② Publish-Subscribe or one to many

## Point-to-Point Messaging



## Publish/Subscribe Messaging



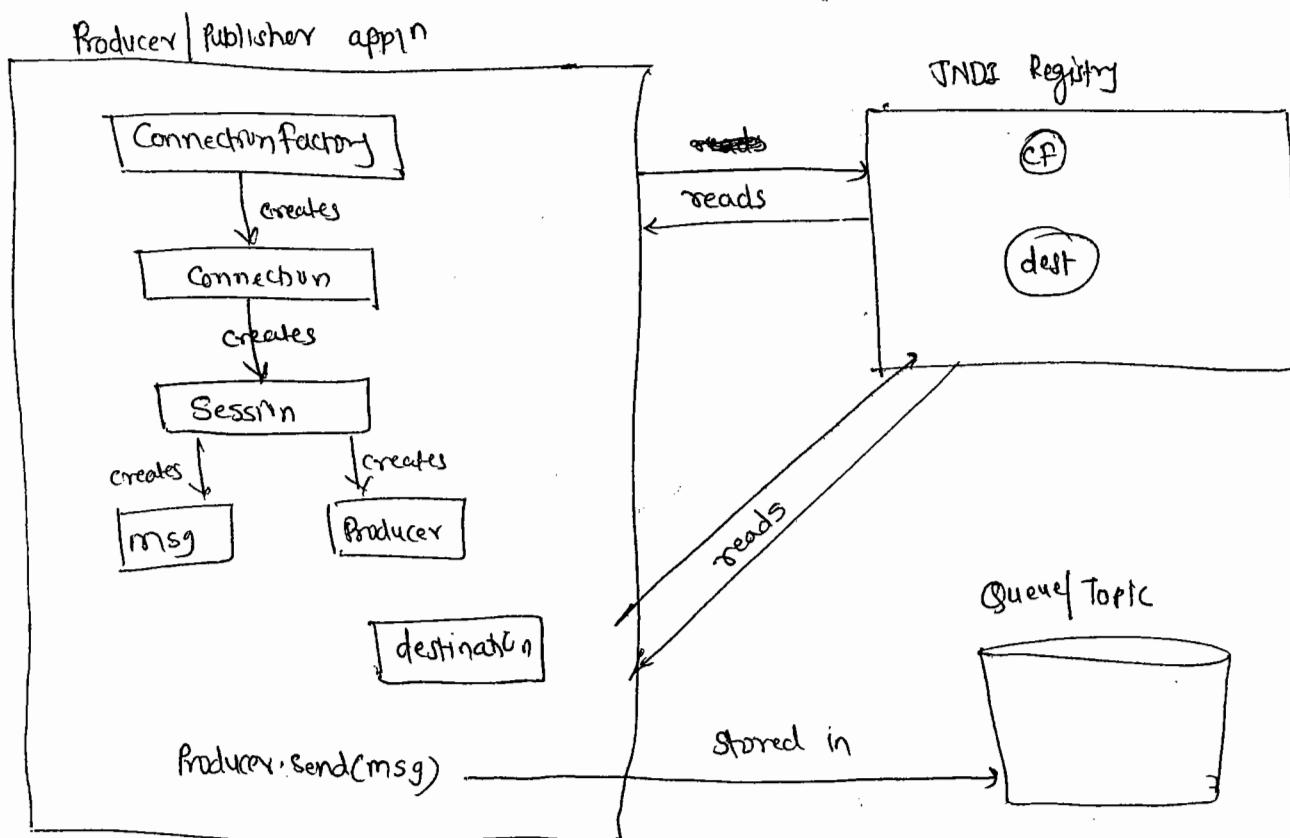
→ In JMS a client application which want to create a message and to send it to destination it should follow the below steps

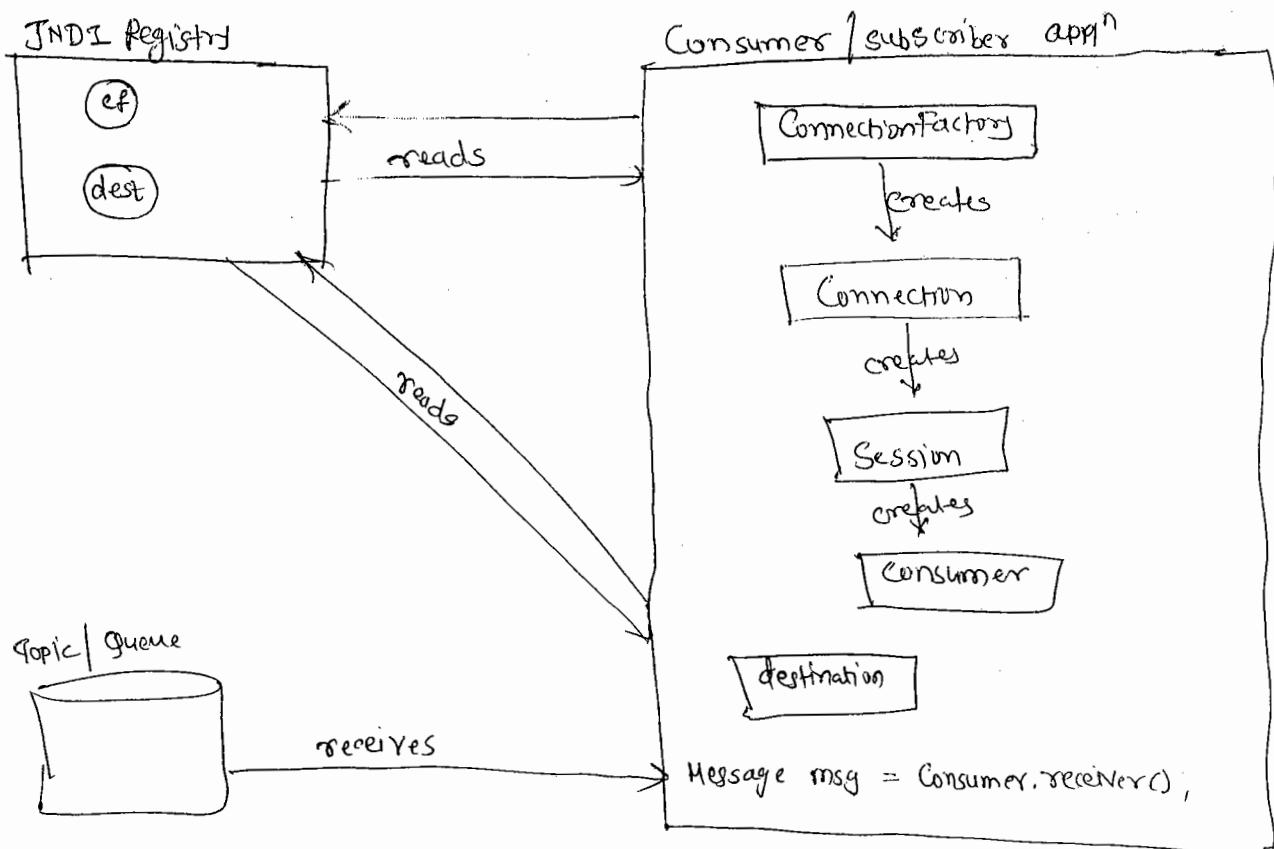
- ① A client application should read a `ConnectionFactory` object from JNDI Registry.
- ② Using `ConnectionFactory` creates a Connection to the JMS Server
- ③ Using Connection a session is created

- ④ Using a session a message & producer object are created.
- ⑤ a client appl<sup>n</sup> reads dest<sup>n</sup> object from <sup>JNDI</sup> registry.
- ⑥ a producer object will send message to destination object.

→ a JMS client appl<sup>n</sup> who wants to receive a message should follow the below steps.

- ① a client application should read a Connectionfactory object from JNDI registry
- ② using Connectionfactory a Connection is created to JMS server
- ③ Using Connection a Session is created
- ④ Using Session a Consumer object is created.
- ⑤ a client application reads destination object from registry
- ⑥ a Consumer object receive a message from destination object.



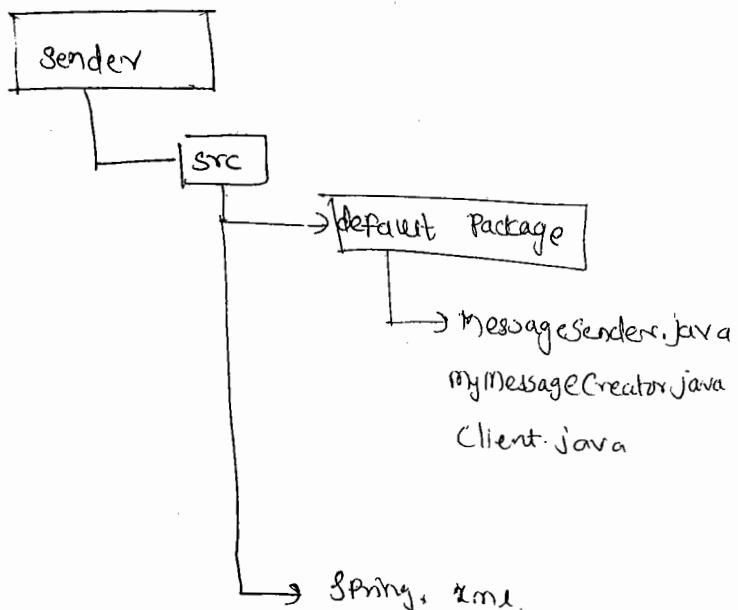


- When developing a JMS appn in Point to point model or in publish subscribe model, there is a BoilerplateCode at both producer & consumer application.
- In order to avoid a BoilerplateCode Spring fw has given Spring-jms as an abstraction layer on top Java jms API
- In a Spring-Jms, JmsTemplate class is given & the using this template class a Spring bean can send a message to a destination message or it can receive a message from destination object
- In JMS 4 types of messages can be send from a producer Appn to a Consumer Appn.
  - ① TextMessage → To send a string as data
  - ② MapMessage → To send a data as key/value pair
  - ③ BytesMessage → To send an image as data
  - ④ ObjectMessage → To send a serialized object of Java as message

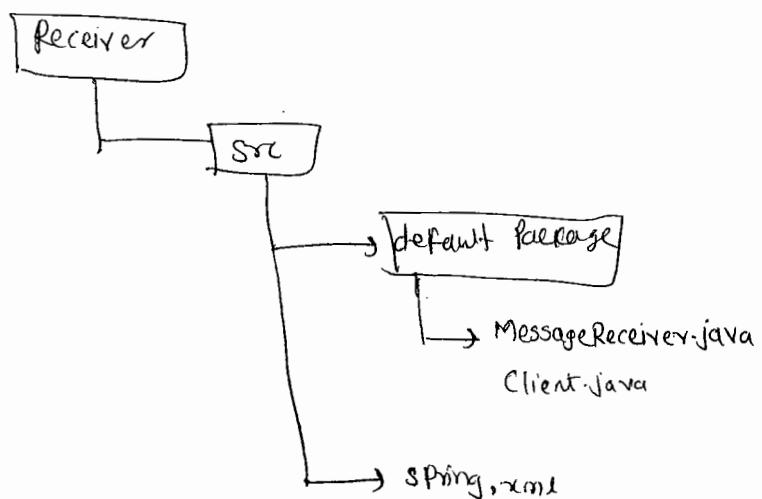
- the above 4 classes are subclasses of an abstract class Message.
- In spring to create a Message object & to set the data framework has given an interface MessageCreator & we need to implement the interface into a class
- Configuring JMS administrative object in Glassfish Server

### Steps

- ① Start the glassfish Server  
 C:\Program Files\glassfish-4.0\glassfish\bin> asadmin start-domain
- ② Open the browser & type the following request.  
 http://localhost:4848
- ③ Expand Resources → select JMS Resources → Click on Connectionfactories  
 Click on new button  
 Enter JNDI Name → qcfactory  
 ResourceType → javax.jms.QueueConnectionfactory → OK
- ④ Click on jms Resources <sup>select</sup> → Destination Resources → new button →  
 Enter JNDI Name → que  
 Physical Destination Name → que  
 ResourceType → javax.jms.Queue → OK



Refer page 6 of Spring Services handout



refer page 7 of spring Services

- ① aopalliance-1.0
- ② appserv-rt.jar → C:\program files\glassfish-4.0\glassfish\lib
- ③ common-logging
- ④ jms-1.1
- ⑤ spring-aop
- ⑥ spring-beans
- ⑦ spring-context
- ⑧ spring-context-support
- ⑨ spring-core-4.0.5
- ⑩ spring-expression
- ⑪ spring-jms
- ⑫ spring-tx



# AOP

```
1 Title: AOP Application with pointcut (1)
2 -----[Demo.java]-----
3 package com.sathya.spring.aop.model;
4 public interface Demo {
5 void sayHello(); pos1
6 void sayBye();
7 String getMessage();
8 void welcome();
9 }
10 -----[DemoBean.java]----- pojo (Business class) | target-class
11 package com.sathya.spring.aop.model;
12 public class DemoBean implements Demo {
13 @Override
14 public String getMessage() {
15 return "I am getMessage";
16 }
17 @Override
18 public void sayBye() {
19 System.out.println("I am sayBye");
20 }
21 @Override
22 public void sayHello() {
23 System.out.println("I am sayHello");
24 }
25 @Override
26 public void welcome() {
27 System.out.println("I am welcome");
28 }
29 } //end of the class
30 -----[WelcomeAdvice.java]-----
31 package com.sathya.spring.aop.aspect;
32 import java.lang.reflect.Method;
33 import org.springframework.aop.MethodBeforeAdvice;
34 public class WelcomeAdvice implements MethodBeforeAdvice {
35 {
36 @Override
37 public void before(Method arg0, Object[] arg1, Object arg2)
38 throws Throwable {
39 String methodName=arg0.getName();
40 System.out.println("I am BeforeAdvice");
41 System.out.println("Welcome to : "+methodName);
42 }
43 }
44 -----[applicationContext.xml]-----
45 <beans xmlns="http://www.springframework.org/schema/beans"
46 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
47 xsi:schemaLocation="http://www.springframework.org/schema/beans
48 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
49
50 <!-- Business class configuration -->
51 <bean id="db" class="com.sathya.spring.aop.model.DemoBean"/>
52 <!-- advice configuration -->
53 <bean id="before" class="com.sathya.spring.aop.aspect>WelcomeAdvice"/>
54 <!-- pointcut configuration -->
55 <bean id="pointcut1" class="org.springframework.aop.support.JdkRegexpMethodPointcut">
56 <property name="patterns">
57 <list>
58 <value>.*say.*</value> the method which satisfies this condition called jointPoint
59 </list>
60 </property>
61 </bean>
62 <!-- advisor configuration -->
63 <bean id="advisor1" class="org.springframework.aop.support.DefaultPointcutAdvisor">
64 <property name="pointcut" ref="pointcut1"/>
65 <property name="advice" ref="before"/> Advisor = Pointcut + advice
66 </bean>
```

## Output

I am BeforeAdvice  
welcome to : sayHello

I am sayHello

=====

I am BeforeAdvice  
welcome to : sayBye

I am sayBye

-----  
I am getMessage

-----  
I am welcome

```

67 <!-- proxy factory bean configuration -->
68 <bean id="pfb1"
69 class="org.springframework.aop.framework.ProxyFactoryBean">
70 <property name="target" ref="db"/> ↗ Weaver class
71 <property name="proxyInterfaces"
72 value="com.sathya.spring.aop.model.Demo"/>
73 <property name="interceptorNames">
74 <list> pass
75 <value>advisor1</value> we need to add advisor class id
76 </list>
77 </property>
78 </bean>
79 </beans>
80 -----AOPMain.java-----
81 //AOPMain.java
82 import org.springframework.context.ApplicationContext;
83 import org.springframework.context.support.ClassPathXmlApplicationContext;
84 import com.sathya.spring.aop.model.Demo;
85 public class AOPMain {
86 public static void main(String[] args) {
87 ApplicationContext ctx =
88 new ClassPathXmlApplicationContext("applicationContext.xml"); } Container created or stored
89 Object o =ctx.getBean("pfb1");
90 Demo demo=(Demo)o;
91 demo.sayHello();
92 System.out.println("=====");
93 demo.sayBye();
94 System.out.println("=====");
95 String str =demo.getMessage();
96 System.out.println(str);
97 System.out.println("=====");
98 demo.welcome();
99 }
100 }
101 =====
102 Title: AOP Application with user-defined Dynamic pointcut (2)
103 -----Factorial.java-----
104 package com.sathya.spring.aop.model;
105 public interface Factorial {
106 void findFactorial(int x);
107 }
108 -----FactorialImpl.java-----
109 package com.sathya.spring.aop.model;
110 public class FactorialImpl implements Factorial
111 {
112 @Override
113 public void findFactorial(int x)
114 {
115 if(x<0)
116 {
117 System.out.println("parameter is -ve");
118 return;
119 }
120 int fact=1;
121 for(int i=1; i<=x; i++)
122 {
123 fact=fact * i;
124 }
125 System.out.println("factorial of :" +x + " is :" +fact);
126 }
127 }
128 -----WelcomeAdvice.java-----
129 package com.sathya.spring.aspect;
130 import java.lang.reflect.Method;
131 import org.springframework.aop.MethodBeforeAdvice;
132 public class WelcomeAdvice implements MethodBeforeAdvice

```

Application context container can do  
Preinstantiation of Singleton class, but not Beanfactory Container.

factorial of : 0 is : 1  
I am BeforeAdvice  
Welcome to : findFactorial  
factorial of : 5 is : 120  
Parameter is -ve

```
133 {
134 @Override
135 public void before(Method arg0, Object[] arg1, Object arg2)
136 throws Throwable {
137 String methodName=arg0.getName();
138 System.out.println("I am BeforeAdvice");
139 System.out.println("Welcome to : "+methodName);
140 }
141 }
142 -----MyDynamicPointcut.java-----
143 package com.sathya.spring.aop.pointcut;
144 import java.lang.reflect.Method;
145 import org.springframework.aop.support.DynamicMethodMatcherPointcut;
146 Import com.sathya.spring.aop.model.FactorialImpl;
147
148 public class MyDynamicPointcut extends DynamicMethodMatcherPointcut
149 {
150 @Override
151 public boolean matches(Method method, Class clazz, Object[] args)
152 {
153 boolean flag=false;
154 int k=(Integer)args[0];
155 If(clazz == FactorialImpl.class)
156 {
157 if(method.getName().startsWith("f") && k>0)
158 {
159 flag=true;
160 }
161 }
162 }
163 return flag;
164 }
165 }
166 -----applicationContext.xml-----
167 <beans xmlns="http://www.springframework.org/schema/beans"
168 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
169 xsi:schemaLocation="http://www.springframework.org/schema/beans
170 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
171
172 <!-- business class configuration -->
173 <bean id="fact"
174 class="com.sathya.spring.aop.model.FactorialImpl"/>
175 <!-- advice configuration -->
176 <bean id="before"
177 class="com.sathya.spring.aop.aspect.WelcomeAdvice"/>
178 <!-- pointcut configuration -->
179 <bean id="pointcut1"
180 class="com.sathya.spring.aop.pointcut.MyDynamicPointcut"/>
181 <!-- advisor configuration -->
182 <bean id="advisor1" class="org.springframework.aop.support.DefaultPointcutAdvisor">
183 <property name="pointcut" ref="pointcut1"/>
184 <property name="advice" ref="before"/>
185 </bean>
186 <!-- proxy factory bean configuration -->
187 <bean id="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
188 <property name="target" ref="fact"/>
189 <property name="proxyInterfaces" value="com.sathya.spring.aop.model.Factorial"/>
190 <property name="interceptorNames">
191 <list>
192 <value>advisor1</value>
193 </list>
194 </property>
195 </bean>
196 </beans>
197 -----AOPMain.java-----
198 import org.springframework.context.ApplicationContext;
```

```

199 import org.springframework.context.support.ClassPathXmlApplicationContext;
200 import com.sathya.spring.aop.model.Factorial;
201 public class AOPMain {
202 public static void main(String args[])
203 {
204 ApplicationContext ctx=
205 new ClassPathXmlApplicationContext("applicationContext.xml");
206 Object o = ctx.getBean("pfb");
207 Factorial f = (Factorial)o;
208 f.findFactorial(0);
209 System.out.println("=====");
210 f.findFactorial(5);
211 System.out.println("=====");
212 f.findFactorial(-1);
213 }
214 }
215 =====
216 Title: AOP Application with XML based AOP
217 -----Product.java-----

```

```

218 package com.sathya.spring.aop.model;
219 public interface Product {
220 void add(int x,int y);
221 int multiply(int x,int y);
222 }
223 -----ProductImpl.java-----
224 package com.sathya.spring.aop.model;
225 public class ProductImpl implements Product {
226 @Override
227 public void add(int x, int y) {
228 System.out.println("Addition = "+(x+y));
229 }
230 @Override
231 public int multiply(int x, int y) {
232 return (x*y);
233 }
234 }

```

-----MyAdvices.java-----

```

235 package com.sathya.spring.aspect;
236 import org.aspectj.lang.JoinPoint;
237 public class MyAdvices {
238 public void before(JoinPoint jp)
239 {
240 String methodName=jp.getSignature().getName();
241 Object args[]=jp.getArgs();
242 System.out.println("I am before advice to : "+methodName);
243 System.out.println("It's arguments values are : "+args[0]+", "+args[1]);
244 }
245 public void afterReturning(JoinPoint jp, Object result)
246 {
247 String methodName=jp.getSignature().getName();
248 Object args[]=jp.getArgs();
249 System.out.println("I am after returning advice to : "+methodName);
250 System.out.println("It's arguments values are : "+args[0]+", "+args[1]);
251 System.out.println("The result of this method is : "+result);
252 }
253 }
254 }

```

-----applicationContext.xml-----

```

255 <!-- applicationContext.xml -->
256 <beans xmlns="http://www.springframework.org/schema/beans"
257 xmlns:aop="http://www.springframework.org/schema/aop"
258 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
259 xsi:schemaLocation="http://www.springframework.org/schema/beans
260 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
261 http://www.springframework.org/schema/aop
262 http://www.springframework.org/schema/aop
263 http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
264 <!-- business class configuration -->

```

O/P

I am before advice to : add

It's arguments values are : 10, 20

Addition = 30

=====

I am after returning advice to : multiply

It's arguments values are : 10, 20

The result of this method is : 200

200

Xmlns → tags for configuring  
beans.

```

265 <bean id="pimpl" class="com.sathya.spring.aop.model.ProductImpl"/>
266 <!-- advices class configuration -->
267 <bean id="ma" class="com.sathya.spring.aop.aspect.MyAdvices"/> target class | Business
268 <!-- xml based aop configuration --> class
269 <aop:config>
270 <aop:aspect ref="ma"> id of Advices class
271 <aop:pointcut id="pt1" interface method
272 expression="execution(* com.sathya.spring.aop.model.Product.add(..))"/>
273 <aop:pointcut id="pt2" execution → pointcut designation
274 expression="execution(* com.sathya.spring.aop.model.Product.multiply(..))"/>
275 <aop:before pointcut-ref="pt1" method="before"/>
276 <aop:after-returning pointcut-ref="pt2"
277 method="afterReturning" returning="result"/>
278 </aop:aspect>
279 </aop:config>
280 <aop:aspectj-autoproxy/> return value of business method store
281 </beans> spring container call Weaver class
282 -----AOPMain.java-----
283 import org.springframework.context.ApplicationContext;
284 import org.springframework.context.support.ClassPathXmlApplicationContext;
285 import com.sathya.spring.aop.model.Product;
286 public class AopMain {
287 public static void main(String[] args) {
288 ApplicationContext ctx=
289 new ClassPathXmlApplicationContext("applicationContext.xml");
290 Object obj=ctx.getBean("pimpl"); id of target | business class
291 Product p=(Product)obj; target class id used by proxy class
292 p.add(10,20); for configuration.
293 System.out.println("=====");
294 p.multiply(10, 20);
295 }
296 }
297 =====
298 Title: AOP Application with Annotations based AOP ④
299 -----Demo.java-----
300 package com.sathya.spring.aop.model;
301 public interface Demo {
302 void businessMethod1(int amount) throws Exception;
303 void businessMethod2();
304 }
305 -----DemoImpl.java-----
306 package com.sathya.spring.aop.model;
307 import com.sathya.spring.aop.exception.InsufficientAmountException;
308 public class DemoImpl implements Demo
309 {
310 @Override
311 public void businessMethod1(int amount) throws Exception
312 {
313 if(amount <=0)
314 throw new InsufficientAmountException("U have given Amount as : "+
315 amount+" it is not valid");
316 else
317 System.out.println("Amount is : "+amount+" it is valid");
318 }
319 @Override
320 public void businessMethod2() {
321 try
322 {
323 Thread.sleep(5000);
324 }
325 catch(Exception e)
326 {
327 }
328 }
329 -----InsufficientAmountException.java-----
330 package com.sathya.spring.aop.exception;

```

```

331 public class InsufficientAmountException extends
332 RuntimeException
333 {
334 public InsufficientAmountException(String message)
335 {
336 super(message);
337 }
338 }
339 -----[MyAspect.java]-----
340 package com.sathya.spring.aop.aspect;
341 import org.aspectj.lang.JoinPoint;
342 import org.aspectj.lang.ProceedingJoinPoint;
343 import org.aspectj.lang.annotation.After;
344 import org.aspectj.lang.annotation.Around;
345 import org.aspectj.lang.annotation.Aspect;
346 @Aspect
347 public class MyAspect {
348 @After("execution(* com.sathya.spring.aop.model.Demo.businessMethod1(..))")
349 public void after(JoinPoint jp)
350 {
351 String str = jp.getSignature().getName();
352 System.out.println("I am after advice to : "+str);
353 }
354 @Around("execution(* com.sathya.spring.aop.model.Demo.businessMethod2(..))")
355 public Object around(ProceedingJoinPoint pjp)
356 {
357 Object o=null;
358 try{
359 long x=System.currentTimeMillis();
360 o = pjp.proceed();
361 long y=System.currentTimeMillis();
362 long z = y-x;
363 String str=pjp.getSignature().getName();
364 System.out.println("I am around advice to : "+str);
365 System.out.println("Time taken to run business is : "+z +" milliseconds");
366 }catch(Throwable t)
367 {
368 return o;
369 }
370 }
371 -----[spring.xml]-----
372 <beans xmlns="http://www.springframework.org/schema/beans"
373 xmlns:aop="http://www.springframework.org/schema/aop"
374 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
375 xsi:schemaLocation="http://www.springframework.org/schema/beans
376 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
377 http://www.springframework.org/schema/aop
378 http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
379 <!-- business class configuration -->
380 <bean id="di" class="com.sathya.spring.aop.model.DemoImpl"/>
381 <!-- aspect class configuration -->
382 <bean id="myaspect" class="com.sathya.spring.aop.aspect.MyAspect"/>
383 <aop:aspectj-autoproxy/>
384 </beans>
385 -----[AOPMain.java]-----
386 import org.springframework.context.ApplicationContext;
387 import org.springframework.context.support.ClassPathXmlApplicationContext;
388 import com.sathya.spring.aop.model.Demo;
389 public class AOPMain
390 {
391 public static void main(String... args)
392 {
393 ApplicationContext ctx=
394 new ClassPathXmlApplicationContext("spring.xml");
395 Object o = ctx.getBean("di");
396 Demo d=(Demo)o;

```

Unchecked exception  
as extends from RuntimeException.

Q How do you find the your Java  
program takes this much time to  
execute program.  
ANS.

```
397 try{
398 d.businessMethod1(1000);
399 }
400 catch(Exception e)
401 {
402 System.out.println(e);
403 }
404 System.out.println("=====");
405 d.businessMethod2();
406 System.out.println("=====");
407 try{
408 d.businessMethod1(-100);
409 }catch(Exception e)
410 {
411 System.out.println(e);
412 }
413 }
414 =====
415 =====
```

DIP Amount is : 1000 it is valid

I am after advice to : businessMethod1

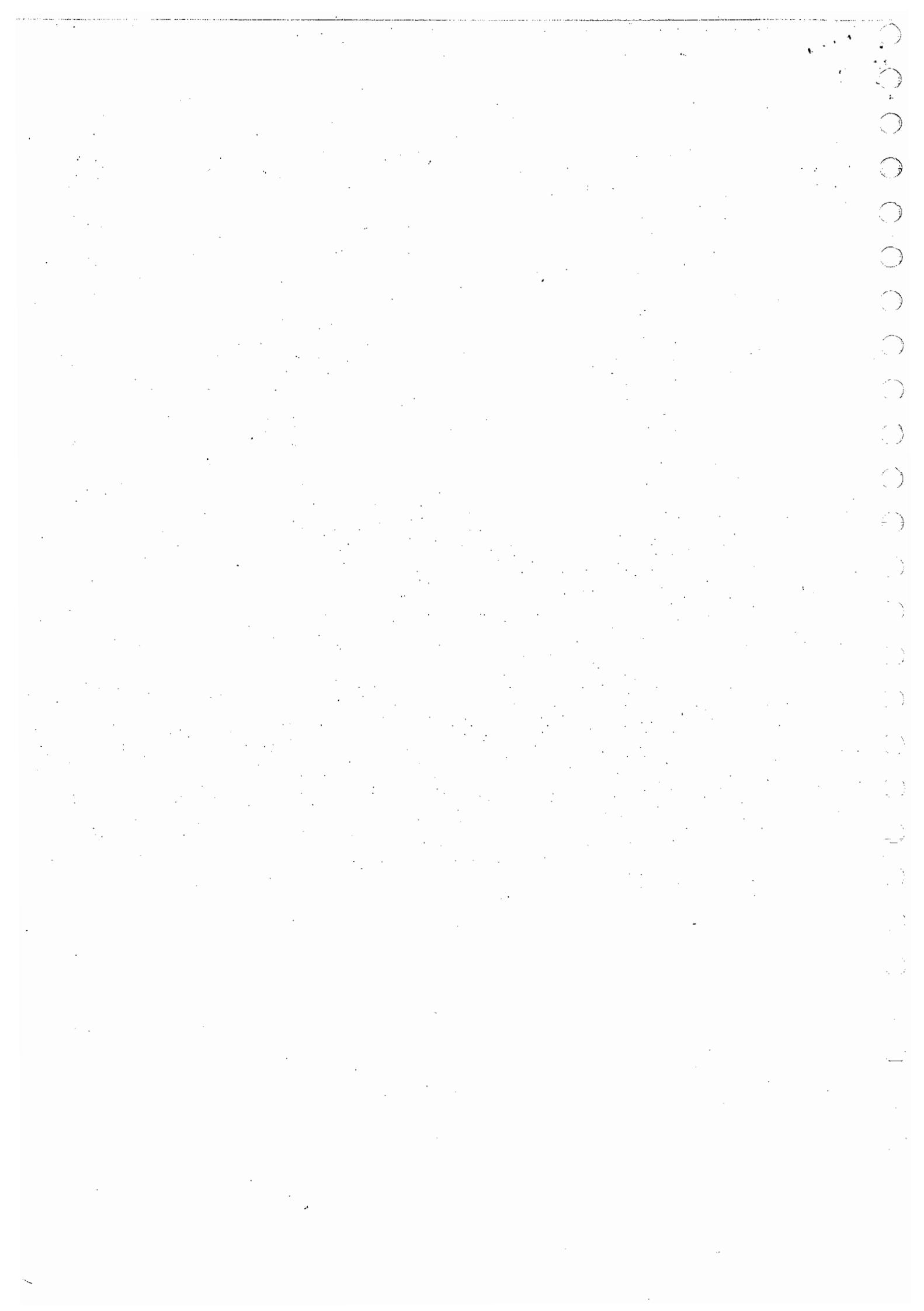
I am around advice to : businessMethod2

Time taken to run business is : 5000 millisecond

I am after advice to : businessMethod1

com.sathya.spring.aop.exception.InsufficientAmountException: U have given

A given amount as -100 It is not valid.



1 Title: Login MVC with Database support using Spring-JDBC  
2 -----  
3 <jsp:forward page="login.form"></jsp:forward>  
4 -----  
5 <form action="check.form" method="post">  
6 Username : <input type=text name="t1"> <br>  
7 Password : <input type=password name="t2"><br>  
8 <input type=submit value="submit">  
9 </form>  
10 -----  
11 <beans xmlns="http://www.springframework.org/schema/beans"  
12 xmlns:context="http://www.springframework.org/schema/context"  
13 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
14 xsi:schemaLocation="http://www.springframework.org/schema/beans  
15 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  
16 http://www.springframework.org/schema/context  
17 http://www.springframework.org/schema/context/spring-context-4.0.xsd">  
18  
19 <context:component-scan base-package="com.sathya.spring.controller"/>  
20 <context:annotation-config/>  
21 <!-- handler mapping -->  
22 <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />  
23 <!-- view resolver -->  
24 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
25 <property name="prefix" value="/" />  
26 <property name="suffix" value=".jsp" />  
27 </bean>  
28  
29 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">  
30 <property name="dataSource" ref="ds"/>  
31 </bean>  
32 <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
33 <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />  
34 <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE" />  
35 <property name="username" value="system" />  
36 <property name="password" value="root" />  
37 </bean>  
38 </beans>  
39 -----  
40 package com.sathya.spring.controller;  
41 import org.springframework.beans.factory.annotation.Autowired;  
42 import org.springframework.stereotype.Controller;  
43 import org.springframework.web.bind.annotation.RequestMapping;  
44 import org.springframework.web.bind.annotation.RequestMethod;  
45 import org.springframework.web.bind.annotation.RequestParam;  
46 import org.springframework.web.servlet.ModelAndView;  
47 @Controller  
48 public class LoginController  
49 {  
50 private LoginService loginService;  
51 @Autowired  
52 public void setLoginService(LoginService loginService)  
53 {  
54 this.loginService=loginService;  
55 }  
56 @RequestMapping(value="/login", method=RequestMethod.GET)  
57 public ModelAndView getLoginPage()  
58 {  
59 return new ModelAndView("login");  
60 }  
61  
62 @RequestMapping(value="check", method=RequestMethod.POST)  
63 public ModelAndView checkLogin(@RequestParam("t1")String s1,@RequestParam("t2")  
64 {  
65 boolean b = loginService.check(s1,s2);  
66 if(b)

```

67 return new ModelAndView("success");
68 else
69 return new ModelAndView("failure");
70 }
71 }
72 -----
73 package com.sathya.spring.controller;
74 import org.springframework.beans.factory.annotation.Autowired;
75 import org.springframework.beans.factory.annotation.Qualifier;
76 import org.springframework.jdbc.core.JdbcTemplate;
77 import org.springframework.stereotype.Repository;
78 @Repository
79 public class LoginService
80 {
81 private JdbcTemplate jt;
82 @Autowired
83 @Qualifier("jt")
84 public void setJt(JdbcTemplate jt)
85 {
86 this.jt=jt;
87 }
88 public boolean check(String s1, String s2)
89 {
90 int i= jt.queryForInt("select count(*) from users where uname=? and pwd=?");
91 if(i==1)
92 return true;
93 else
94 return false;
95 }
96 }
97 -----
98 <h1> Login Success </h1> My name: ${uname} or Hi : ${name}
99 -----
100 <h1> Login Failed </h1>
101 =====
102 Title: Login MVC with annotation validations
103 [index.jsp] from DispatcherServlet forward request
104 <jsp:forward page="loginpage.form"></jsp:forward> to Login Page
105 [login.jsp]
106 <%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
107 <form:form method="post" action="check.form" commandName="loginFormBackingObject">
108 Username: <form:input path="uname"/> <form:errors path="uname"/>

109 Password : <form:password path="pwd"/><form:errors path="pwd"/>

110 <input type=submit value="click"/> bind Pwd element to POJO class property Pwd
111 </form:form> binding If it is a property name of POJO class
112 -----spring-servlet.xml----- Key of POJO class object
113 <beans xmlns="http://www.springframework.org/schema/beans"
114 xmlns:mvc="http://www.springframework.org/schema/mvc"
115 xmlns:context="http://www.springframework.org/schema/context"
116 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
117 xsi:schemaLocation="http://www.springframework.org/schema/beans
118 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
119 http://www.springframework.org/schema/context
120 http://www.springframework.org/schema/context/spring-context-4.0.xsd
121 http://www.springframework.org/schema/mvc
122 http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
123
124 <context:component-scan base-package="com.sathya.spring.mvc"/>
125 <!-- view resolver -->
126 <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceView"
127 <property name="prefix" value="/" />
128 <property name="suffix" value=".jsp"/>
129 </bean>
130 <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessage
131 <property name="basename" value="messages"/>
132 </bean>

```

Resource bundle name

```

133 <mvc:annotation-driven/> → to enable the validation of annotations.
134 </beans>
135 ----- Login.java -----
136 //Login.java (condmna class)
137 package com.sathya.spring.mvc;
138
139 import javax.validation.constraints.Size;
140
141 import org.hibernate.validator.constraints.NotEmpty;
142
143 public class Login
144 {
145 @Size(min=4,max=8)
146 private String uname;
147 @NotEmpty
148 private String pwd;
149
150 //setters & getters
151 public String getUsername() {
152 return uname;
153 }
154 public void setUsername(String uname) {
155 this.uname = uname;
156 }
157
158 public String getPassword() {
159 return pwd;
160 }
161
162 public void setPassword(String pwd) {
163 this.pwd = pwd;
164 }
165 }

166 ----- LoginController.java -----
167 package com.sathya.spring.mvc;
168 import javax.validation.Valid;
169 import org.springframework.stereotype.Controller;
170 import org.springframework.validation.BindingResult;
171 import org.springframework.web.bind.annotation.ModelAttribute;
172 import org.springframework.web.bind.annotation.RequestMapping;
173 import org.springframework.web.bind.annotation.RequestMethod;
174 import org.springframework.web.servlet.ModelAndView;
175 @Controller
176 public class LoginController "/ " is optional
177 {
178 @RequestMapping(value="loginpage",method=RequestMethod.GET)
179 public ModelAndView getLoginPage()
180 {
181 return new ModelAndView("login","loginFormBackingObject",new Login());
182 } ↓ view name ↓ key to link b/w Login form & object of Pojo class
183
184 @RequestMapping(value="check",method=RequestMethod.POST)
185 public ModelAndView processRequest(@Valid @ModelAttribute("loginFormBackingObject")↓
186 Login login, BindingResult result)
187 {
188 if(result.hasErrors()) ↳ It stores error → to check error are there or not & apply if else cond
189 { ↳ object
190 if(true if there are errors) ↳ in result object
191 { ↳ key of pojo class object
192 return new ModelAndView("login");
193 }
194 String username=login.getUsername();
195 String password=login.getPassword();
196 if(username.equals("sathya") && password.equals("java"))
197 return new ModelAndView("success");
198 else
199 }
200 }

```

**@Valid** → It is for executing the validations

```

199 return new ModelAndView("failure");
200 }
201 }
202 }
203 -----messages.properties-----
204 #messages.properties
205 Size.loginFormBackingObject.uname=Username should be from {2} to {1} length
206 NotEmpty.loginFormBackingObject.pwd=Password is required
207 -----failure.jsp-----
208 <h1>
209 <%
210 out.println("Login failed");
211 %>
212 </h1>
213 -----success.jsp-----
214 <h1>
215 <%
216 out.println("Login Success");
217 %>
218 </h1>
219 =====
220
221 Title: FileUpload Application
222 =====
223 -----FileUploadController.java-----
224 package com.sathya.spring.mvc.controller;
225 import java.io.FileNotFoundException;
226 import java.io.FileOutputStream;
227 import java.io.IOException;
228 import javax.servlet.ServletContext;
229 import javax.servlet.http.HttpServletRequest;
230 import javax.servlet.http.HttpSession;
231 import org.springframework.stereotype.Controller;
232 import org.springframework.web.bind.annotation.RequestMapping;
233 import org.springframework.web.bind.annotation.RequestMethod;
234 import org.springframework.web.bind.annotation.RequestParam;
235 import org.springframework.web.multipart.MultipartFile;
236 import org.springframework.web.servlet.ModelAndView;
237 @Controller
238 public class FileUploadController
239 {
240 @RequestMapping(value="upload", method=RequestMethod.POST)
241 public ModelAndView fileUpload(@RequestParam("file1")
242 MultipartFile file, HttpServletRequest request)
243 {
244 String fail="";
245 String success="";
246 if(file.getSize()!=0)
247 {
248 if(file.getContentType().equals("text/xml"))
249 {
250 try {
251 FileOutputStream fos=new FileOutputStream("E:/"+file.getOriginalFilename());
252 fos.write(file.getBytes());
253 success="U r file : "+file.getOriginalFilename()+" Uploaded Successfully";
254 } catch (FileNotFoundException e) {
255 // TODO Auto-generated catch block
256 e.printStackTrace();
257 } catch (IOException e) {
258 // TODO Auto-generated catch block
259 e.printStackTrace();
260 }
261 }
262 }
263 }
264 fail="The file should be XML only!!!";

```

the Parameters are replaced in  
alphabetical order so {1} is Max &  
{2} is Min

Min                          max value.  
4                          8

size → name of annotation

```
265 }
266 } //outer if
267 else
268 {
269 fail="The file should not be empty!!!";
270 }
271 if(success.length()>1)
272 {
273 return new ModelAndView("success","message",success);
274 }
275 else
276 {
277 return new ModelAndView("uploadfile","message",fail);
278 }
279 } //end of fileUpload() method
280 }//end of class
281 -----success.jsp-----
282 <h1>
283 ${message}
284 </h1>
285 -----uploadfile.jsp-----
286 <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
287 <html>
288 <body>
289 <center>
290 <c:if test="${message ne null}">
291 <c:out value="${message}" />
292 </c:if>
293

294 <form action="upload.html" method="post" enctype="multipart/form-data">
295 Select a File : <input type=file name="file1">

296 <input type=submit value="UPLOAD">
297 </form>
298 </center>
299 </body>
300 </html>
301 -----spring-servlet.xml-----
302 <beans xmlns="http://www.springframework.org/schema/beans"
303 xmlns:context="http://www.springframework.org/schema/context"
304 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
305 xsi:schemaLocation="http://www.springframework.org/schema/beans
306 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
307 http://www.springframework.org/schema/context
308 http://www.springframework.org/schema/context/spring-context-3.0.xsd">
309 <context:component-scan base-package="com.sathya.spring.mvc.controller"/>
310 <bean id="viewResolver"
311 class="org.springframework.web.servlet.view.InternalResourceViewResolver">
312 <property name="prefix" value="/" />
313 <property name="suffix" value=".jsp" />
314 </bean>
315 <bean id="multipartResolver"
316 class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>
317 </beans>
318 -----web.xml-----
319 <?xml version="1.0" encoding="UTF-8"?>
320 <web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
321 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
322 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
323 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
324 <display-name>
325 SpringMVC</display-name>
326 <servlet>
327 <servlet-name>spring</servlet-name>
328 <servlet-class>
329 org.springframework.web.servlet.DispatcherServlet</servlet-class>
330 <load-on-startup>1</load-on-startup>
```

```
331 </servlet>
332 <servlet-mapping>
333 <servlet-name>spring</servlet-name>
334 <url-pattern>*.html</url-pattern>
335 </servlet-mapping>
336 </web-app>
337 Title: SpringI18N application
338 =====
339 -----MyController.java-----
340 package com.sathya.spring.i18n;
341 import org.springframework.stereotype.Controller;
342 import org.springframework.web.bind.annotation.RequestMapping;
343 import org.springframework.web.servlet.ModelAndView;
344 @Controller
345 public class MyController
346 {
347 @RequestMapping("i18ntest")
348 public ModelAndView welcome()
349 {
350 return new ModelAndView("index");
351 }
352 }
353 -----index.jsp-----
354 <!-- index.jsp -->
355 <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
356 <%@page session="false" %>
357 <html>
358 <body>
359 English |
360 Telugu |
361 Hindi
362

363

364 <center>
365 <spring:message code="welcome.msg" text="This is default text" />
366 </center>
367 </body>
368 </html>
369 -----messages_en.properties-----
370 #english
371 welcome.msg=This is a message written in English
372 -----messages_te.properties-----
373 # telugu
374 welcome.msg=ఈ సందేశంతెుగులొవ్రాసి
376 ది
377 -----messages_hi.properties-----
378 # hindi
379 welcome.msg=यह एक् सन्दश् हिन्द् मे िख हे
382 -----spring-servlet.xml-----
383 <beans xmlns="http://www.springframework.org/schema/beans"
384 xmlns:mvc="http://www.springframework.org/schema/mvc"
385 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
386 xmlns:context="http://www.springframework.org/schema/context"
387 xsi:schemaLocation="http://www.springframework.org/schema/beans
388 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
389 http://www.springframework.org/schema/context
390 http://www.springframework.org/schema/context/spring-context-4.0.xsd
391 http://www.springframework.org/schema/mvc
392 http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
393 <context:component-scan
394 base-package="com.sathya.spring.i18n" />
395 <!-- Application Message Bundle -->
396 <bean id="messageSource"
```

```
397 class="org.springframework.context.support.ResourceBundleMessageSource">
398 <property name="basename" value="messages" />
399 </bean>
400 <mvc:interceptors>
401 <bean id="localeChange"
402 class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
403 <property name="paramName" value="lang" />
404 </bean>
405 </mvc:interceptors>
406 <bean id="localeResolver"
407 class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
408 </bean>
409 <bean id="viewResolver"
410 class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
411 <property name="prefix" value="/" />
412 <property name="suffix" value=".jsp"/>
413 </bean>
414 </beans>
415 -----web.xml-----
416 <web-app>
417 <display-name>Spring3MVC</display-name>
418 <welcome-file-list>
419 <welcome-file>index.jsp</welcome-file>
420 </welcome-file-list>
421 <servlet>
422 <servlet-name>spring</servlet-name>
423 <servlet-class>
424 org.springframework.web.servlet.DispatcherServlet
425 </servlet-class>
426 <load-on-startup>1</load-on-startup>
427 </servlet>
428 <servlet-mapping>
429 <servlet-name>spring</servlet-name>
430 <url-pattern>*.form</url-pattern>
431 </servlet-mapping>
432 <listener>
433 <listener-class>org.springframework.web.context.ContextLoaderListener
434 </listener-class>
435 </listener>
436 <context-param>
437 <param-name>contextConfigLocation</param-name>
438 <param-value>/WEB-INF/spring-servlet.xml</param-value>
439 </context-param>
440 </web-app>
441 -----
```

(5)

442 Title: SpringLoginI18N application

```
443 =====
444 -----Login.java-----
445 //command class
446 //Login.java
447 package com.sathya.spring.mvc.controller;
448 import javax.validation.constraints.Size;
449 import org.hibernate.validator.constraints.NotEmpty;
450 public class Login {
451 @NotEmpty
452 private String uname;
453 @NotEmpty
454 private String pwd;
455 public String getUserName() {
456 return uname;
457 }
458 public void setUserName(String uname) {
459 this.uname = uname;
460 }
461 public String getPassword() {
462 return pwd;
```

```

463 }
464 public void setPwd(String pwd) {
465 this.pwd = pwd;
466 }
467 }
468 -----LoginController.java-----
469 package com.sathya.spring.mvc.controller;
470 import javax.validation.Valid;
471 import org.springframework.stereotype.Controller;
472 import org.springframework.validation.BindingResult;
473 import org.springframework.web.bind.annotation.ModelAttribute;
474 import org.springframework.web.bind.annotation.RequestMapping;
475 import org.springframework.web.bind.annotation.RequestMethod;
476 import org.springframework.web.bind.annotation.SessionAttributes;
477 import org.springframework.web.servlet.ModelAndView;
478 @Controller
479 public class LoginController {
480 @RequestMapping("loginpage")
481 public ModelAndView getLoginPage()
482 {
483 return new ModelAndView("login","command",new Login());
484 }
485 @RequestMapping(value="logincheck",method=RequestMethod.POST)
486 public ModelAndView verifyLogin(@Valid @ModelAttribute("command")
487 Login login, BindingResult result)
488 {
489 if(result.hasErrors())
490 return new ModelAndView("login");
491 else
492 return new ModelAndView("success","msg","Login Success");
493 }
494 }
495 -----messages_de.properties----- de → german lang. code
496 label.uname=User
497 label.pwd=Pword
498 NotEmpty.command.uname=User is mandatory
499 NotEmpty.command.pwd=Pword is mandatory
500 -----messages_en.properties-----
501 label.uname=Username
502 label.pwd=Password
503 NotEmpty.command.uname=Username is mandatory
504 NotEmpty.command.pwd=Password is mandatory
505 -----spring-servlet.xml-----
506 <beans xmlns="http://www.springframework.org/schema/beans"
507 xmlns:context="http://www.springframework.org/schema/context"
508 xmlns:mvc="http://www.springframework.org/schema/mvc"
509 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
510 xsi:schemaLocation="http://www.springframework.org/schema/beans
511 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
512 http://www.springframework.org/schema/context
513 http://www.springframework.org/schema/context/spring-context-3.0.xsd
514 http://www.springframework.org/schema/mvc
515 http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
516 <mvc:annotation-driven/>
517 <context:component-scan base-package="com.sathya.spring.mvc.controller"/>
518 <bean id="viewResolver"
519 class="org.springframework.web.servlet.view.InternalResourceViewResolver">
520 <property name="prefix" value="/" />
521 <property name="suffix" value=".jsp" />
522 </bean>
523 <bean id="messageSource"
524 class="org.springframework.context.support.ResourceBundleMessageSource">
525 <property name="basename" value="messages" />
526 </bean>
527 <mvc:interceptors>
528 <bean id="localeChangeInterceptor">

```

529        class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">  
530            <property name="paramName" value="language" />  
531        </bean>  
532     </mvc:interceptors>  
533     <bean id="localeResolver"  
534        class="org.springframework.web.servlet.i18n.SessionLocaleResolver"/>  
535     </beans>  
536 -----index.jsp-----  
537 <jsp:forward page="loginpage.html"/>  
538 -----login.jsp-----  
539 <%@ page session="false" %>  
540 <%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>  
541 <%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>  
542 <html>  
543     <body>  
544        <center>  
545            <a href="loginpage.html?language=en">English</a>  
546            <a href="loginpage.html?language=de">German</a> <br>  
547        <form:form action="logincheck.html" method="post">  
548            <spring:message code="label.uname"/> :  
549            <form:input path="uname"/> <form:errors path="uname"/><br>  
550            <spring:message code="label.pwd"/> :  
551            <form:password path="pwd"/><form:errors path="pwd"/> <br>  
552            <input type=submit value="submit">  
553        </form:form>  
554     </body>  
555 </html>  
556 -----success.jsp-----  
557 <%-- success.jsp --%>  
558 <h1>  
559     \${msg}  
560 </h1>  
561 -----failure.jsp-----  
562 <!-- failure.jsp -->  
563 <h1>  
564     \${msg}  
565 </h1>  
566        Title: SpringMVCTiles application  
567 ======  
568 -----TilesController.java-----  
569 package com.sathya.spring.mvc.controller;  
570 import org.springframework.stereotype.Controller;  
571 import org.springframework.web.bind.annotation.RequestMapping;  
572 import org.springframework.web.servlet.ModelAndView;  
573 @Controller  
574 public class TilesController {  
575        @RequestMapping("welcome")  
576        public ModelAndView m1()  
577        {  
578            return new ModelAndView("welcome");  
579        }  
580        @RequestMapping("friends")  
581        public ModelAndView m2()  
582        {  
583            return new ModelAndView("friends");  
584        }  
585        @RequestMapping("office")  
586        public ModelAndView m3()  
587        {  
588            return new ModelAndView("office");  
589        }  
590 }  
591 -----index.jsp-----  
592 <%-- index.jsp --%>  
593 <jsp:forward page="welcome.html"/>  
594 -----welcome.jsp-----

```
595 <h2>
596 Welcome to Spring with Tiles
597 </h2>
598 -----header.jsp-----
599 <div align="center" style="font-weight:bold">
600 Sathya Technologies
601 </div>
602 -----footer.jsp-----
603 <div align="center">© Sekhar.com</div>
604 -----menu.jsp-----
605

606 Friends

607 Office
608 -----layout.jsp-----
609 <!-- layout.jsp -->
610 <%@taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
611 <html!>
612 <body>
613 <table border="1" cellpadding="2" cellspacing="2" align="center">
614 <tr>
615 <td height="30" colspan="2">
616 <tiles:insertAttribute name="header" />
617 </td>
618 </tr>
619 <tr>
620 <td height="250">
621 <tiles:insertAttribute name="menu" />
622 </td>
623 <td width="350">
624 <tiles:insertAttribute name="body" />
625 </td>
626 </tr>
627 <tr>
628 <td height="30" colspan="2">
629 <tiles:insertAttribute name="footer" />
630 </td>
631 </tr>
632 </table>
633 </body>
634 </html>
635 -----office.jsp-----
636 <html>
637 <body>
638 <center>
639 SATHYA TECHNOLOGIES

640 AMEERPET

641 HYD
642 </center>
643 </body>
644 </html>
645 -----friends.jsp-----
646 <html>
647 <body>
648 <center>
649 STRUTS

650 SPRING

651 HIBERNATE

652 </center>
653 </body>
654 </html>
655 -----spring-servlet.xml-----
656 <beans xmlns="http://www.springframework.org/schema/beans"
657 xmlns:context="http://www.springframework.org/schema/context"
658 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
659 xsi:schemaLocation="http://www.springframework.org/schema/beans
660 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
```

```

661 http://www.springframework.org/schema/context
662 http://www.springframework.org/schema/context/spring-context-3.0.xsd">
663 <context:component-scan base-package="com.sathya.spring.mvc.controller"/>
664 <bean id="viewResolver"
665 class="org.springframework.web.servlet.view.UrlBasedViewResolver">
666 <property name="viewClass"
667 value="org.springframework.web.servlet.view.tiles2.TilesView"/>
668 </bean>
669 <bean id="configurer"
670 class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
671 <property name="definitions">
672 <list>
673 <value>/WEB-INF/tiles.xml</value>
674 </list>
675 </property>
676 </bean>
677 </beans>
678 -----tiles.xml-----
679 <!DOCTYPE tiles-definitions PUBLIC
680 "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
681 "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">
682 <!-- tiles.xml -->
683 <tiles-definitions>
684 <definition name="baseLayout" template="/layout.jsp">
685 <put-attribute name="header" value="/header.jsp"/>
686 <put-attribute name="menu" value="/menu.jsp"/>
687 <put-attribute name="body" value="" />
688 <put-attribute name="footer" value="/footer.jsp"/>
689 </definition>
690
691 <definition name="welcome" extends="baseLayout">
692 <put-attribute name="body" value="/welcome.jsp"/>
693 </definition>
694
695 <definition name="friends" extends="baseLayout">
696 <put-attribute name="body" value="/friends.jsp"/>
697 </definition>
698
699 <definition name="office" extends="baseLayout">
700 <put-attribute name="body" value="/office.jsp"/>
701 </definition>
702 </tiles-definitions>
703 -----web.xml-----
704 <?xml version="1.0" encoding="UTF-8"?>
705 <web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
706 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
707 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/i2ee/web-app_2_4.xsd">
708 <display-name>
709 SpringMVC</display-name>
710 <servlet>
711 <servlet-name>spring</servlet-name>
712 <servlet-class>org.springframework.web.servlet.DispatcherServlet
713 </servlet-class>
714 <load-on-startup>1</load-on-startup>
715 </servlet>
716 <servlet-mapping>
717 <servlet-name>spring</servlet-name>
718 <url-pattern>*.html</url-pattern>
719 </servlet-mapping>
720 </web-app>
721
722
723 Title: PdfExcelApp application
724 =====
725 -----MyController.java-----
726 package com.sathya.spring;

```

```
727 import org.springframework.stereotype.Controller;
728 import org.springframework.web.bind.annotation.RequestMapping;
729 import org.springframework.web.servlet.ModelAndView;
730 @Controller
731 public class MyController
732 {
733 @RequestMapping("pdf")
734 public ModelAndView pdfResponse()
735 {
736 return new ModelAndView("pdf");
737 }
738 @RequestMapping("excel")
739 public ModelAndView excelResponse()
740 {
741 return new ModelAndView("excel");
742 }
743 }
744 }
745 -----MyExcelView.java-----
746 package com.sathya.spring;
747 import java.util.Map;
748 import javax.servlet.http.HttpServletRequest;
749 import javax.servlet.http.HttpServletResponse;
750 import org.apache.poi.hssf.usermodel.HSSFRichTextString;
751 import org.apache.poi.hssf.usermodel.HSSFSheet;
752 import org.apache.poi.hssf.usermodel.HSSFWorkbook;
753 import org.springframework.web.servlet.view.document.AbstractExcelView;
754 public class MyExcelView extends AbstractExcelView {
 @Override
 protected void buildExcelDocument(Map arg0,HSSFWorkbook arg1,
 HttpServletRequest arg2, HttpServletResponse arg3)
 throws Exception {
 HSSFSheet sheet=arg1.createSheet("Names");
 getCell(sheet,0,0).setCellValue(new HSSFRichTextString("User Names"));
 getCell(sheet,1,0).setCellValue(new HSSFRichTextString("Sathya"));
 getCell(sheet,2,0).setCellValue(new HSSFRichTextString("Ram"));
 getCell(sheet,3,0).setCellValue(new HSSFRichTextString("Rom"));
 }
766 }
767 -----MyPdfView.java-----
768 package com.sathya.spring;
769 import java.util.Map;
770 import javax.servlet.http.HttpServletRequest;
771 import javax.servlet.http.HttpServletResponse;
772 import org.springframework.web.servlet.view.document.AbstractPdfView;
773 import com.lowagie.text.Document;
774 import com.lowagie.text.Paragraph;
775 import com.lowagie.text.Table;
776 import com.lowagie.text.pdf.PdfWriter;
777 public class MyPdfView extends AbstractPdfView
778 {
779 @Override
780 protected void buildPdfDocument(Map arg0, Document arg1,
781 PdfWriter arg2, HttpServletRequest arg3, HttpServletResponse arg4)
782 throws Exception {
783 Paragraph p = new Paragraph("User details");
784 p.setAlignment("center");
785 arg1.add(p);
786 Table t = new Table(1);
787 t.addCell("Sathya");
788 t.addCell("Ram");
789 t.addCell("Rom");
790 arg1.add(t);
791 }
792 }
```

HSSF (Horrible SPreadsheet format)

```

793 -----index.html-----
794 <center>
795 PDF

796 Excel
797 </center>
798
799 -----spring-servlet.xml-----
800 <beans xmlns="http://www.springframework.org/schema/beans"
801 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
802 xmlns:context="http://www.springframework.org/schema/context"
803 xsi:schemaLocation="http://www.springframework.org/schema/beans
804 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
805 http://www.springframework.org/schema/context
806 http://www.springframework.org/schema/context/spring-context-3.0.xsd">
807 <context:component-scan
808 base-package="com.sathya.spring" />
809 <bean id="id"
810 class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
811 <bean name="pdf" class="com.sathya.spring.MyPdfView"/>
812 <bean name="excel" class="com.sathya.spring.MyExcelView"/>
813 </beans>
814
815 -----web.xml-----
816 <web-app>
817 <display-name>Spring3MVC</display-name>
818 <welcome-file-list>
819 <welcome-file>index.html</welcome-file>
820 </welcome-file-list>
821 <servlet>
822 <servlet-name>spring</servlet-name>
823 <servlet-class>
824 org.springframework.web.servlet.DispatcherServlet
825 </servlet-class>
826 <load-on-startup>1</load-on-startup>
827 </servlet>
828 <servlet-mapping>
829 <servlet-name>spring</servlet-name>
830 <url-pattern>*.form</url-pattern>
831 </servlet-mapping>
832
833 </web-app>
834

```

(8) 835 Title: struts and spring integration application

```

836 =====
837 -----login.jsp-----
838 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
839 <center>
840 <html:errors/> → to display the error messages
841 <html:form action="login.do">
842 Username : <html:text property="uname"/>
843

844 Password: <html:password property="pwd"/>
845

846 <html:submit value="click"/>
847 </html:form>
848 </center>
849 -----web.xml-----
850 <web-app>
851 <servlet>
852 <servlet-name>action</servlet-name>
853 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
854 <load-on-startup> 1 </load-on-startup>
855 <init-param>
856 <param-name>config</param-name>
857 <param-value>/WEB-INF/struts-config.xml</param-value>
858 </init-param>

```

It is front controller  
of Struts

It is a struts configuration file

```

859 </servlet>
860 <servlet-mapping>
861 <servlet-name>action</servlet-name>
862 <url-pattern>*.do</url-pattern>
863 </servlet-mapping>
864 </web-app>
865 -----struts-config.xml-----
866 <!DOCTYPE struts-config PUBLIC
867 "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
868 "http://struts.apache.org/dtds/struts-config_1_3.dtd">
869 <struts-config>
870 <form-beans>
871 <form-bean name="frm1" type="com.sathya.struts.LoginForm"/>
872 </form-beans>
873 <action-mappings>
874 <action path="/login" name="frm1"
875 type="org.springframework.web.struts.DelegatingActionProxy" validate="true">
876 input="/login.jsp"> → this page is forwarded if validation fails ↳ It is to enable to validate
877 <forward name="success" path="/success.jsp"/>
878 <forward name="failure" path="/failure.jsp"/>
879 </action>
880 </action-mappings>
881 <message-resources parameter="ApplicationResources"/> → it is resource bundle name
882 <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
883 <set-property property="contextConfigLocation"
884 value="/WEB-INF/applicationContext.xml"/> → spring-configuration file.
885 </plug-in>
886 </struts-config>
887 -----LoginForm.java-----
888 package com.sathya.struts;
889 import org.apache.struts.action.*;
890 import javax.servlet.http.*;
891 public class LoginForm extends ActionForm
892 {
893 private String uname, pwd;
894
895 public void setUname(String uname)
896 {
897 this.uname = uname;
898 }
899 public String getUname()
900 {
901 return uname;
902 }
903
904 public void setPwd(String pwd)
905 {
906 this.pwd = pwd;
907 }
908 public String getPwd()
909 {
910 return pwd;
911 }
912 public ActionErrors validate(ActionMapping mapping,
913 HttpServletRequest request)
914 {
915 ActionErrors aes = new ActionErrors();
916 if(uname.equals("") || uname.length() == 0)
917 {
918 ActionMessage am1 = new ActionMessage("user.wrong");
919 aes.add("uname", am1);
920 }
921 if(pwd.equals("") || pwd.length() == 0)
922 {
923 ActionMessage am2 = new ActionMessage("pwd.wrong");
924 aes.add("pwd", am2);
925 }
926 }
927 }

```

both are same

URL pattern of request

→ It is given by spring file

It is key in the Resource bundle.

```
925 }
926 return aes;
927 }
928 }
929 -----LoginAction.java-----
930 package com.sathya.struts;
931 import javax.servlet.http.*;
932 import org.apache.struts.action.*;
933 import com.sathya.spring.model.LoginBean;
934 public class LoginAction extends Action
935 {
936 //Here LoginBean is a spring bean. LoginAction
937 //depends on LoginBean
938 private LoginBean loginBean;
939 //setter injection
940 public void setLoginBean(LoginBean loginBean)
941 {
942 this.loginBean=loginBean;
943 }
944 public ActionForward execute(ActionMapping mapping,ActionForm form,
945 HttpServletRequest req,HttpServletResponse res) throws Exception
946 {
947 LoginForm lf=(LoginForm)form;
948 String s1=lf.getUname();
949 String s2=lf.getPwd();
950 //call business method of spring bean
951 int k = loginBean.verify(s1,s2);
952 if(k==1)
953 return mapping.findForward("success");
954 else
955 return mapping.findForward("failure");
956 }
957 }//execute
958 } //class
959 -----LoginBean.java-----
960 package com.sathya.spring.model;
961 import com.sathya.spring.dao.LoginDAO;
962 public class LoginBean {
963 private LoginDAO loginDAO;
964 public void setLoginDAO(LoginDAO loginDAO)
965 {
966 this.loginDAO=loginDAO;
967 }
968 public int verify(String str1,String str2)
969 {
970 int k=loginDAO.check(str1, str2);
971 return k;
972 }
973 }
974 -----LoginDAO.java-----
975 package com.sathya.spring.dao;
976 public interface LoginDAO {
977 int check(String str1,String str2);
978 }
979 -----LoginDAOImpl.java-----
980 package com.sathya.spring.dao;
981 import org.springframework.jdbc.core.JdbcTemplate;
982 public class LoginDAOImpl implements LoginDAO {
983 private JdbcTemplate jt;
984 public void setJt(JdbcTemplate jt)
985 {
986 this.jt=jt;
987 }
988 @Override
989 public int check(String str1, String str2)
990 {
```

```
991 int k=jt.queryForInt("select count(*) from login_table
992 where username=? and password=?",str1,str2);
993 return k;
994 }
995 }
996 -----applicationContext.xml-----
997 <beans xmlns="http://www.springframework.org/schema/beans"
998 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
999 xsi:schemaLocation="http://www.springframework.org/schema/beans
1000 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
1001 <!-- struts action class configuration -->
1002 <bean name="/login" class="com.sathya.struts.LoginAction">
1003 <property name="loginBean" ref="logBean"/>
1004 </bean>
1005 <!-- LoginBean configuration -->
1006 <bean id="logBean" class="com.sathya.spring.model.LoginBean">
1007 <property name="loginDAO" ref="logDAO"/>
1008 </bean>
1009 <!-- dao configuration -->
1010 <bean id="logDAO" class="com.sathya.spring.dao.LoginDAOImpl">
1011 <property name="jt" ref="jt"/>
1012 </bean>
1013 <!-- JdbcTemplate-configuration -->
1014 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
1015 <property name="dataSource" ref="ds"/>
1016 </bean>
1017 <!-- DataSource configuration -->
1018 <bean id="ds"
1019 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
1020 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
1021 <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl3"/>
1022 <property name="username" value="system"/>
1023 <property name="password" value="tiger"/>
1024 </bean>
1025 </beans>
1026 -----success.jsp-----
1027 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
1028 <h2>
1029 U entered Username as : <bean:write name="frm1" property="uname"/>

1030 U entered Password as : <bean:write name="frm1" property="pwd"/>

1031 </h2>
1032 <%
1033 out.println("<h2> login success </h2>");
1034 %>
1035 -----failure.jsp-----
1036 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
1037 <h2>
1038 U entered Username as : <bean:write name="frm1" property="uname"/>

1039 U entered Password as : <bean:write name="frm1" property="pwd"/>

1040 </h2>
1041 <%
1042 out.println("<h2> login failed </h2>");
1043 %>
1044 -----ApplicationResources.properties-----
1045 user.wrong=Username is required

1046 pwd.wrong=>Password is required
1047
```

```
1 =====
2 Title: Spring-JDBC-Example1
3 -----Customer.java-----
4 package com.sathya.spring.dto;
5 public class Customer {
6 private int customerId;
7 private String customerName;
8 private String customerAddress;
9
10 // --> setters & getters <--
11 public int getCustomerId() {
12 return customerId;
13 }
14 public void setCustomerId(int customerId) {
15 this.customerId = customerId;
16 }
17 public String getCustomerName() {
18 return customerName;
19 }
20 public void setCustomerName(String customerName) {
21 this.customerName = customerName;
22 }
23 public String getCustomerAddress() {
24 return customerAddress;
25 }
26 public void setCustomerAddress(String customerAddress) {
27 this.customerAddress = customerAddress;
28 }
29 }
30 -----CustomerDAO.java-----
31 //CustomerDAO.java
32 //DAO interface
33 package com.sathya.spring.dao;
34 import com.sathya.spring.dto.Customer;
35 public interface CustomerDAO {
36 int insert(Customer c);
37 }
38 -----CustomerDAOImpl.java-----
39 //DAO class
40 package com.sathya.spring.dao;
41 import org.springframework.jdbc.core.JdbcTemplate;
42 import com.sathya.spring.dto.Customer;
43 public class CustomerDAOImpl implements CustomerDAO {
44 private JdbcTemplate jt;
45 //setter injection
46 public void setJt(JdbcTemplate jt)
47 {
48 this.jt=jt;
49 }
50 @Override
51 public int insert(Customer c) {
52 int cid=c.getCustomerId();
53 String cname=c.getCustomerName();
54 String caddr=c.getCustomerAddress();
55 int k=jt.update("insert into customer_table values(?,?,?)",
56 cid,cname,caddr);
57 return k;
58 }
59 }
60 -----DemoBean.java-----
61 //Business class
62 package com.sathya.spring.business;
63 import com.sathya.spring.dao.CustomerDAO;
64 import com.sathya.spring.dto.Customer;
65 public class DemoBean {
66 private CustomerDAO customerDAO;
```

```
67 //setter injection
68 public void setCustomerDAO(CustomerDAO customerDAO)
69 {
70 this.customerDAO=customerDAO;
71 }
72 public void insertCustomer(int cid, String cname, String caddr)
73 {
74 Customer c=new Customer();
75 c.setCustomerId(cid);
76 c.setCustomerName(cname);
77 c.setCustomerAddress(caddr);
78 int k=customerDAO.insert(c);
79 System.out.println(k+" row inserted");
80 }
81 }
-----applicationContext.xml-----
82 <!-- applicationContext.xml -->
83 <beans xmlns="http://www.springframework.org/schema/beans"
84 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
85 xsi:schemaLocation="http://www.springframework.org/schema/beans
86 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
87 <!-- business class configuration -->
88 <bean id="db" class="com.sathya.spring.business.DemoBean">
89 <property name="customerDAO" ref="customerDAO"/>
90 </bean>
91 <!-- DAO class configuration -->
92 <bean id="customerDAO" class="com.sathya.spring.dao.CustomerDAOImpl">
93 <property name="jt" ref="jt"/>
94 </bean>
95 <!-- JdbcTemplate class configuration -->
96 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
97 <constructor-arg ref="ds"/>
98 </bean>
99 <!-- DataSource bean configuration -->
100 <bean id="ds"
101 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
102 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
103 <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
104 <property name="username" value="system"/>
105 <property name="password" value="tiger"/>
106 </bean>
107 </beans>
108 -----
109 -----Main.java-----
110 import org.springframework.context.ApplicationContext;
111 import org.springframework.context.support.ClassPathXmlApplicationContext;
112 import com.sathya.spring.business.DemoBean;
113 public class Main {
114 public static void main(String[] args) {
115 ApplicationContext ctx=new ClassPathXmlApplicationContext
116 ("applicationContext.xml");
117 Object o=ctx.getBean("db");
118 DemoBean db=(DemoBean)o;
119 db.insertCustomer(4, "D", "Hyd");
120 }
121 }
=====
```

② 123 Title: Spring-JDBC-Example2

```
124 -----DemoBean.java-----
125 package com.sathya.spring.business;
126 import java.util.Iterator;
127 import java.util.List;
128 import java.util.Map;
129 import java.util.Set;
130 import org.springframework.beans.factory.annotation.Autowired;
131 import org.springframework.beans.factory.annotation.Qualifier;
132 import com.sathya.spring.dao.SelectDAO;
```

```
133 public class DemoBean
134 {
135 private SelectDAO selectDAO;
136 //setter injection
137 @Autowired
138 @Qualifier("dao")
139 public void setSelectDAO(SelectDAO selectDAO)
140 {
141 this.selectDAO=selectDAO;
142 }
143 //business method-1
144 public void findCustomerById(int customerId)
145 {
146 Map map=selectDAO.findById(customerId);
147 Set set=map.entrySet();
148 Iterator it=set.iterator();
149 while(it.hasNext())
150 {
151 Map.Entry me=(Map.Entry)it.next();
152 System.out.println(me.getKey()+" : "+me.getValue());
153 }
154 } //end of findCustomerById
155 //Business method-2
156 public void findAllCustomers()
157 {
158 List list=selectDAO.findAll();
159 Iterator it =list.iterator();
160 while(it.hasNext())
161 {
162 Map map=(Map)it.next();
163 Set set=map.entrySet();
164 Iterator it1=set.iterator();
165 while(it1.hasNext())
166 {
167 Map.Entry me=(Map.Entry)it1.next();
168 System.out.println(me.getKey()+" : "+me.getValue());
169 } //end of inner while loop
170 System.out.println("=====");
171 } //end of outer while loop
172 } //end of findAllCustomers
173 }
174 -----
175 -----SelectDAO.java-----
176 //SelectDAO
177 //DAO interface
178 package com.sathya.spring.dao;
179 import java.util.List;
180 import java.util.Map;
181 public interface SelectDAO {
182 Map findById(int custid);
183 List findAll();
184 }
185 -----
186 -----SelectDAOImpl.java-----
187 //SelectDAOImpl.java
188 //DAO class
189 package com.sathya.spring.dao;
190 import java.util.List;
191 import java.util.Map;
192 import org.springframework.beans.factory.annotation.Autowired;
193 import org.springframework.beans.factory.annotation.Qualifier;
194 import org.springframework.jdbc.core.JdbcTemplate;
195 public class SelectDAOImpl implements SelectDAO {
196 private JdbcTemplate jt;
197 //setter injection
198 @Autowired
199 @Qualifier("jt")
```

```

199 public void setJt(JdbcTemplate jt)
200 {
201 this.jt=jt;
202 }
203 @Override
204 public List findAll() {
205 List list=jt.queryForList("select * from customer_table");
206 return list;
207 }
208 @Override
209 public Map findById(int custid) {
210 Map map=jt.queryForMap("select * from customer_table where cid=?",custid);
211 return map;
212 }
213 }

214 -----applicationContext.xml-----
215 <beans xmlns="http://www.springframework.org/schema/beans"
216 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
217 xsi:schemaLocation="http://www.springframework.org/schema/beans
218 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
219 <!-- business class configuration -->
220 <bean id="db" class="com.sathya.spring.business.DemoBean"/>
221 <!-- dao class configuration -->
222 <bean id="dao" class="com.sathya.spring.dao.SelectDAOImpl"/>
223 <!-- JdbcTemplate class configuration -->
224 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
225 <property name="dataSource" ref="ds"/>
226 </bean>
227 <!-- DataSource class configuration -->
228 <bean id="ds" class="org.apache.tomcat.dbcp.dbcp.BasicDataSource">
229 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
230 <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl3"/>
231 <property name="username" value="system"/>
232 <property name="password" value="tiger"/>
233 </bean>
234 <context:annotation-config/> autowired to enable to the annotation.
235 </beans>
236 -----Main.java-----
237 import org.springframework.context.ApplicationContext;
238 import org.springframework.context.support.ClassPathXmlApplicationContext;
239 import com.sathya.spring.business.DemoBean;
240 public class Main {
241 public static void main(String args[])
242 {
243 ApplicationContext ctx=new ClassPathXmlApplicationContext
244 ("applicationContext.xml");
245 Object o=ctx.getBean("db");
246 DemoBean db=(DemoBean)o;
247 db.findCustomerById(1);
248 System.out.println("=====");
249 db.findAllCustomers();
250 }
251 }
252 =====
253 ③ Title: Spring-JDBC-Example3 (with RowMapper)
254 -----DemoBean.java-----
255 //DemoBean.java(business class)
256 package com.sathya.spring.model;
257 import java.util.Iterator;
258 import java.util.List;
259 import com.sathya.spring.dao.EmployeeDAO;
260 public class DemoBean {
261 private EmployeeDAO employeeDAO;
262 //setter injection
263 public void setEmployeeDAO(EmployeeDAO employeeDAO)
264 {

```

```
265 this.employeeDAO=employeeDAO;
266 }
267 public void findAllEmployees()
268 {
269 List list=employeeDAO.selectAllEmployees();
270 Iterator it= list.iterator();
271 while(it.hasNext())
272 {
273 Employee e=(Employee)it.next();
274 System.out.println("empno = "+e.getEmpno());
275 System.out.println("ename = "+e.getEname());
276 System.out.println("sal = "+e.getSal());
277 System.out.println("deptno = "+e.getDeptno());
278 System.out.println("=====");
279 }
280 }
281 }
282 -----Employee.java-----
283 //Employee.java (pojo)
284 package com.sathya.spring.model;
285 public class Employee {
286 private int empno;
287 private String ename;
288 private int sal;
289 private int deptno;
290
291 //setters & getters
292 public int getEmpno() {
293 return empno;
294 }
295 public void setEmpno(int empno) {
296 this.empno = empno;
297 }
298 public String getEname() {
299 return ename;
300 }
301 public void setEname(String ename) {
302 this.ename = ename;
303 }
304 public int getSal() {
305 return sal;
306 }
307 public void setSal(int sal) {
308 this.sal = sal;
309 }
310 public int getDeptno() {
311 return deptno;
312 }
313 public void setDeptno(int deptno) {
314 this.deptno = deptno;
315 }
316 }
317 -----EmployeeDAO.java-----
318 //EmployeeDAO.java (Dao interface)
319 package com.sathya.spring.dao;
320 import java.util.List;
321 public interface EmployeeDAO {
322 List selectAllEmployees();
323 }
324 -----EmployeeDAOImpl.java-----
325 //EmployeeDAOImpl.java(dao class)
326 package com.sathya.spring.dao;
327 import java.util.List;
328 import org.springframework.jdbc.core.JdbcTemplate;
329 public class EmployeeDAOImpl implements EmployeeDAO {
330 private JdbcTemplate jt;
```

```

331 //setter injection
332 public void setJt(JdbcTemplate jt)
333 {
334 this.jt=jt;
335 }
336 @Override
337 public List selectAllEmployees() {
338 *→ List list=jt.query("select * from emp",new EmployeeMapper());
339 return list; ↓ Command ↓ mapRow();
340 }
341 }
-----EmployeeMapper.java-----
342 //EmployeeMapper.java(callback class)
343 package com.sathya.spring.dao;
344 import java.sql.ResultSet;
345 import java.sql.SQLException;
346 import org.springframework.jdbc.core.RowMapper;
347 import com.sathya.spring.model.Employee;
348 public class EmployeeMapper implements RowMapper
349 {
350 ↗ callback interface
351 @Override
352 public Object mapRow(ResultSet rs, int index) throws SQLException
353 {
354 Employee e=new Employee();
355 e.setEmpno(rs.getInt(1));
356 e.setEname(rs.getString(2));
357 e.setSal(rs.getInt(3));
358 e.setDeptno(rs.getInt(4));
359 return e;
360 }
361 }
-----applicationContext.xml-----
362 <beans xmlns="http://www.springframework.org/schema/beans"
363 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
364 xsi:schemaLocation="http://www.springframework.org/schema/beans
365 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
366
367 <!-- business class configuration -->
368 <bean id="db" class="com.sathya.spring.model.DemoBean">
369 <property name="employeeDAO" ref="dao"/>
370 </bean>
371 <!-- dao class configuration -->
372 <bean id="dao" class="com.sathya.spring.dao.EmployeeDAOImpl">
373 <property name="jt" ref="jt"/>
374 </bean>
375 <!-- JdbcTemplate class configuration -->
376 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
377 <property name="dataSource" ref="ds"/>
378 </bean>
379 <!-- DataSource configuration -->
380 <bean id="ds"
381 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
382 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
383 <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl3"/>
384 <property name="username" value="system"/>
385 <property name="password" value="tiger"/>
386 </bean>
387 </beans>
388 -----Main.java-----
389 import org.springframework.context.ApplicationContext;
390 import org.springframework.context.support.ClassPathXmlApplicationContext;
391 import com.sathya.spring.model.DemoBean;
392 public class Main {
393 public static void main(String[] args) {
394 ApplicationContext ctx = new ClassPathXmlApplicationContext
395 ("applicationContext.xml");
396 Object o = ctx.getBean("db");

```

Callback interface  
used to define a part of code  
to logic by program  
to JdbcTemplate

DemoBean depends on  
EmployeeDAOImpl

EmployeeDAOImpl  
depends on JdbcTemplate

JdbcTemplate depends  
on DataSource.

```
397 DemoBean db=(DemoBean)o;
398 db.findAllEmployees();
399 }
400 }
401 =====
402 ② Title: Spring-JDBC-Example4(for calling a function) ④
403 -----DemoBean.java-----
404 package com.sathya.spring.model;
405 import java.util.Map;
406 import org.springframework.beans.factory.annotation.Autowired;
407 import org.springframework.beans.factory.annotation.Qualifier;
408 import com.sathya.spring.dao.TestDAO;
409 public class DemoBean
410 {
411 private TestDAO testDAO;
412 @Autowired
413 @Qualifier("dao")
414 public void setTestDAO(TestDAO testDAO)
415 {
416 this.testDAO=testDAO;
417 }
418 public void bm()
419 {
420 Map map=testDAO.callFunction();
421 int bonus=(Integer)map.get("b");
422 System.out.println("Bonus = "+bonus);
423 }
424 }
425 -----TestDAO.java-----
426 //TestDAO.java(dao interface)
427 package com.sathya.spring.dao;
428 import java.util.Map;
429 public interface TestDAO {
430 Map callFunction();
431 }
432 -----TestDAOImpl.java-----
433 //TestDAOImpl.java (dao class)
434 package com.sathya.spring.dao;
435 import java.sql.Types;
436 import java.util.ArrayList;
437 import java.util.Map;
438 import org.springframework.beans.factory.annotation.Autowired;
439 import org.springframework.beans.factory.annotation.Qualifier;
440 import org.springframework.jdbc.core.JdbcTemplate;
441 import org.springframework.jdbc.core.SqlOutParameter;
442 import org.springframework.jdbc.core.SqlParameter;
443 public class TestDAOImpl implements TestDAO
444 {
445 private JdbcTemplate jt;
446 @Autowired
447 @Qualifier("jtemplate")
448 public void setJt(JdbcTemplate jt)
449 {
450 this.jt=jt;
451 }
452 @Override
453 public Map callFunction()
454 {
455 ArrayList alist=new ArrayList();
456 alist.add(new SqlOutParameter("b", Types.INTEGER));
457 alist.add(new SqlParameter(Types.INTEGER));
458 Map map=jt.call(new MyCallableStatementCreator(),alist);
459 return map;
460 }
461 }
462 -----MyCallableStatementCreator.java-----
```

```

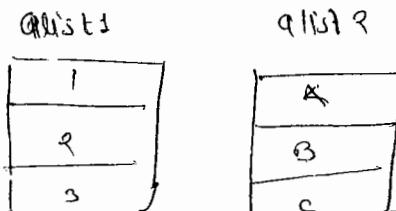
463 package com.sathya.spring.dao;
464 import java.sql.CallableStatement;
465 import java.sql.Connection;
466 import java.sql.SQLException;
467 import java.sql.Types;
468 import org.springframework.jdbc.core.CallableStatementCreator;
469 public class MyCallableStatementCreator implements CallableStatementCreator
470 {
 @Override
 public CallableStatement createCallableStatement(Connection con)
 throws SQLException
 {
 CallableStatement cstmt= con.prepareCall("{?=call fun1(?)}");
 cstmt.setInt(2,7965);
 cstmt.registerOutParameter(1,Types.INTEGER);
 return cstmt;
 }
480 }
-----applicationContext.xml-----
482 <beans xmlns="http://www.springframework.org/schema/beans"
483 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
484 xmlns:context="http://www.springframework.org/schema/context"
485 xsi:schemaLocation="http://www.springframework.org/schema/beans
486 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
487 http://www.springframework.org/schema/context
488 http://www.springframework.org/schema/context/spring-context-3.0.xsd">
489 <!-- business class configuration -->
490 <bean id="db" class="com.sathya.spring.model.DemoBean"/>
491 <!-- dao class configuration -->
492 <bean id="dao" class="com.sathya.spring.dao.TestDAOImpl"/>
493 <!-- JdbcTemplate class configuration -->
494 <bean id="jtemplate"
495 class="org.springframework.jdbc.core.JdbcTemplate">
496 <property name="dataSource" ref="ds"/>
497 </bean>
498 <!-- DataSource configuration -->
499 <bean id="ds"
500 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
501 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
502 <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl3"/>
503 <property name="username" value="system"/>
504 <property name="password" value="tiger"/>
505 </bean>
506 <context:annotation-config/>
507 </beans>
508 -----Main.java-----
509 import org.springframework.context.ApplicationContext;
510 import org.springframework.context.support.ClassPathXmlApplicationContext;
511 import com.sathya.spring.model.DemoBean;
512 public class Main
513 {
514 public static void main(String[] args)
515 {
516 ApplicationContext ctx=new ClassPathXmlApplicationContext
517 ("applicationContext.xml");
518 Object o = ctx.getBean("db");
519 DemoBean db=(DemoBean)o;
520 db.bm();
521 }
522 }
=====5=====
523 =====5=====
524 Title: Spring-JDBC-Batch processing
525 -----DemoBean.java-----5
526 package com.sathya.spring.model;
527 import org.springframework.beans.factory.annotation.Autowired;
528 import org.springframework.beans.factory.annotation.Qualifier;

```

```

529 import com.sathya.spring.dao.TestDAO;
530 public class DemoBean
531 {
532 private TestDAO testDAO;
533 @Autowired
534 @Qualifier("dao")
535 public void setTestDAO(TestDAO testDAO)
536 {
537 this.testDAO=testDAO;
538 }
539 public void bm()
540 {
541 testDAO.insertBatch();
542 }
543 }
544 -----TestDAO.java-----
545 package com.sathya.spring.dao;
546 public interface TestDAO {
547 void insertBatch();
548 }
549 -----TestDAOImpl.java-----
550 package com.sathya.spring.dao;
551 import org.springframework.beans.factory.annotation.Autowired;
552 import org.springframework.beans.factory.annotation.Qualifier;
553 import org.springframework.jdbc.core.JdbcTemplate;
554 public class TestDAOImpl implements TestDAO {
555 private JdbcTemplate jt;
556 @Autowired
557 @Qualifier("jt")
558 public void setJt(JdbcTemplate jt)
559 {
560 this.jt=jt;
561 }
562 @Override
563 public void insertBatch()
564 {
565 int k[]=jt.batchUpdate("insert into Test values(?,?)",
566 new MyBatchPreparedStatementSetter());
567 System.out.println("Batch Executed");
568 }
569 }
570 -----MyBatchPreparedStatementSetter.java-----
571 package com.sathya.spring.dao;
572 import java.sql.PreparedStatement;
573 import java.sql.SQLException;
574 import java.util.ArrayList;
575 import org.springframework.jdbc.core.BatchPreparedStatementSetter;
576 public class MyBatchPreparedStatementSetter implements
577 BatchPreparedStatementSetter {
578 ArrayList alist1 = new ArrayList();
579 ArrayList alist2 = new ArrayList();
580 public MyBatchPreparedStatementSetter()
581 {
582 alist1.add(1);
583 alist1.add(2);
584 alist1.add(3);
585 alist2.add("A");
586 alist2.add("B");
587 alist2.add("C");
588 }
589 @Override
590 public int getBatchSize() {
591 return 3;
592 }
593 @Override
594 public void setValues(PreparedStatement pstmt, int i) throws SQLException

```



```

595 {
596 pstmt.setInt(1, Integer.parseInt(alist1.get(i).toString()));
597 pstmt.setString(2, alist2.get(i).toString());
598 }
599 }
600 -----spring.xml-----
601 <beans xmlns="http://www.springframework.org/schema/beans"
602 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
603 xmlns:context="http://www.springframework.org/schema/context"
604 xsi:schemaLocation="http://www.springframework.org/schema/beans
605 http://www.springframework.org/schema/spring-beans-3.0.xsd
606 http://www.springframework.org/schema/context
607 http://www.springframework.org/schema/context/spring-context-3.0.xsd">
608 <!-- business class configuration -->
609 <bean id="db" class="com.sathya.spring.model.DemoBean"/>
610 <!-- dao class configuration -->
611 <bean id="dao" class="com.sathya.spring.dao.TestDAOImpl"/>
612 <!-- JdbcTemplate class configuration -->
613 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
614 <property name="dataSource" ref="jndi" />
615 </bean>
616 <bean id="jndi" class="org.springframework.jndi.JndiObjectFactoryBean">
617 <property name="jndiName" value="oraclejndi"/>
618 <property name="jndiEnvironment">
619 <props>
620 <prop key="java.naming.factory.initial">
621 com.sun.enterprise.naming.impl.SerialInitContextFactory</prop>
622 <prop key="java.naming.provider.url">iop://localhost:4848</prop>
623 </props>
624 </property>
625 </bean>
626 <context:annotation-config/>
627 </beans>
628 -----Main.java-----
629 import org.springframework.context.ApplicationContext;
630 import org.springframework.context.support.ClassPathXmlApplicationContext;
631 import com.sathya.spring.model.DemoBean;
632 public class Main
633 {
634 public static void main(String args[])
635 {
636 ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
637 Object o = ctx.getBean("db");
638 DemoBean db=(DemoBean)o;
639 db.bm();
640 }
641 }
642 =====
643 Title: Spring-Hibernate integration(with Spring-ORM)
644 -----DemoBean.java-----
645 package com.sathya.spring.model;
646 import java.util.Iterator;
647 import java.util.List;
648 import com.sathya.spring.dao.CurdDAO;
649 public class DemoBean
650 {
651 private CurdDAO curdDAO;
652 //setter injection
653 public void setCurdDAO(CurdDAO curdDAO)
654 {
655 this.curdDAO=curdDAO;
656 }
657 public void insertStudent(int sid, String sname, int marks)
658 {
659 Student s =new Student();
660 s.setStudentId(sid);

```

Configure  
driver manager  
dataSource  
(DriverManagerDataSource)

```
661 s.setStudentName(sname);
662 s.setMarks(marks);
663 curdDAO.save(s);
664 }
665 public void updateStudent(int sid, String sname, int marks)
666 {
667 Student s = new Student();
668 s.setStudentId(sid);
669 s.setStudentName(sname);
670 s.setMarks(marks);
671 curdDAO.update(s);
672 }
673 public void selectStudents()
674 {
675 List list = curdDAO.findStudents();
676 Iterator it = list.iterator();
677 while(it.hasNext())
678 {
679 Student s = (Student)it.next();
680 System.out.println(s.getStudentId() + " " + s.getStudentName()
681 + " " + s.getMarks());
682 } //end of while
683 } //end of method
684 } //end of class
685 -----Student.java-----
686 //Student.java(pojo class)
687 package com.sathya.spring.model;
688 public class Student
689 {
690 private int studentId;
691 private String studentName;
692 private int marks;
693 //setters & getters
694 public int getStudentId() {
695 return studentId;
696 }
697 public void setStudentId(int studentId) {
698 this.studentId = studentId;
699 }
700 public String getStudentName() {
701 return studentName;
702 }
703 public void setStudentName(String studentName) {
704 this.studentName = studentName;
705 }
706 public int getMarks() {
707 return marks;
708 }
709 public void setMarks(int marks) {
710 this.marks = marks;
711 }
712 }
713 -----student.hbm.xml-----
714 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
715 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
716 <!-- student.hbm.xml -->
717 <hibernate-mapping>
718 <class name="com.sathya.spring.model.Student" table="student">
719 <id name="studentId" column="sid"/>
720 <property name="studentName" column="sname"/>
721 <property name="marks" column="marks"/>
722 </class>
723 </hibernate-mapping>
724 -----applicationContext.xml-----
725 <beans xmlns="http://www.springframework.org/schema/beans"
726 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
727 xsi:schemaLocation="http://www.springframework.org/schema/beans
728 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
729 <!-- business class configuration -->
730 <bean id="db" class="com.sathya.spring.model.DemoBean">
731 <property name="curdDAO" ref="dao"/>
732 </bean>
733 <!-- dao class configuration -->
734 <bean id="dao" class="com.sathya.spring.dao.CurdDAOImpl">
735 <property name="ht" ref="ht"/>
736 </bean>
737 <!-- HibernateTemplate class configuration -->
738 <bean id="ht" class="org.springframework.orm.hibernate3.HibernateTemplate">
739 <property name="sessionFactory" ref="lsfb"/>
740 </bean>
741 <!-- LocalSessionFactoryBean configuration -->
742 <bean id="lsfb" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
743 <property name="dataSource" ref="ds"/>
744 <property name="hibernateProperties">
745 <props>
746 <prop key="dialect">org.hibernate.dialect.Oracle10gDialect</prop>
747 <prop key="show_sql">true</prop>
748 </props>
749 </property>
750 <property name="mappingResources">
751 <list>
752 <value> com/sathya/spring/config/student.hbm.xml </value>
753 </list>
754 </property>
755 </bean>
756 <!-- DataSource configuration -->
757 <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
758 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
759 <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl3"/>
760 <property name="username" value="system"/>
761 <property name="password" value="tiger"/>
762 </bean>
763 </beans>
764 -----CurdDAO.java-----
765 //CurdDAO.java
766 //dao interface
767 package com.sathya.spring.dao;
768 import java.util.List;
769 public interface CurdDAO
770 {
771 void save(Object o);
772 void update(Object o);
773 List findStudents();
774 }
775 -----CurdDAOImpl.java-----
776 //CurdDAOImpl.java
777 //dao class
778 package com.sathya.spring.dao;
779 import java.util.List;
780 import org.springframework.orm.hibernate3.HibernateTemplate;
781 public class CurdDAOImpl implements CurdDAO
782 {
783 private HibernateTemplate ht;
784 //setter injection
785 public void setHt(HibernateTemplate ht)
786 {
787 this.ht=ht;
788 }
789
790 @Override
791 public List findStudents() {
792 List list = ht.find("from Student s");
```

```
793 return list;
794 }
795 @Override
796 public void save(Object o) {
797 ht.save(o);
798 }
799 @Override
800 public void update(Object o) {
801 ht.update(o);
802 }
803 }
804 =====Main.java=====
805 //Main.java
806 import org.springframework.context.ApplicationContext;
807 import org.springframework.context.support.ClassPathXmlApplicationContext;
808 import com.sathya.spring.model.DemoBean;
809 public class Main
810 {
811 public static void main(String[] args)
812 {
813 ApplicationContext ctx=
814 new ClassPathXmlApplicationContext(
815 "com/sathya/spring/config/applicationContext.xml");
816 Object o = ctx.getBean("db");
817 DemoBean db=(DemoBean)o;
818 db.insertStudent(111, "AAA", 600);
819 db.updateStudent(111,"BBB", 900);
820 db.selectStudents();
821 }
822 }
823 =====Title: Spring-Hibernate integration(without Spring-ORM)=====
824 =====DemoBean.java=====
825 //DemoBean.java
826 //business class
827 package com.sathya.spring.model;
828 import java.util.Iterator;
829 import java.util.List;
830 import com.sathya.spring.dao.CurdDAO;
831 public class DemoBean
832 {
833 private CurdDAO curdDAO;
834 //setter injection
835 public void setCurdDAO(CurdDAO curdDAO)
836 {
837 this.curdDAO=curdDAO;
838 }
839 public void insertStudent(int sid,String sname,int marks)
840 {
841 Student s =new Student();
842 s.setStudentId(sid);
843 s.setStudentName(sname);
844 s.setMarks(marks);
845 curdDAO.save(s);
846 }
847 public void updateStudent(int sid,String sname,int marks)
848 {
849 Student s =new Student();
850 s.setStudentId(sid);
851 s.setStudentName(sname);
852 s.setMarks(marks);
853 curdDAO.update(s);
854 }
855 public void selectStudents()
856 {
857 List list=curdDAO.findStudents();
```

```
859 Iterator it =list.iterator();
860 while(it.hasNext())
861 {
862 Student s=(Student)it.next();
863 System.out.println(s.getStudentId()+" "+s.getStudentName()
864 +" "+s.getMarks());
865 } //end of while
866 } //end of method
867 } //end of class
868 -----Student.java-----
869 //Student.java(pojo class)
870 package com.sathya.spring.model;
871 public class Student
872 {
873 private int studentId;
874 private String studentName;
875 private int marks;
876 //setters & getters
877 public int getStudentId() {
878 return studentId;
879 }
880 public void setStudentId(int studentId) {
881 this.studentId = studentId;
882 }
883 public String getStudentName() {
884 return studentName;
885 }
886 public void setStudentName(String studentName) {
887 this.studentName = studentName;
888 }
889 public int getMarks() {
890 return marks;
891 }
892 public void setMarks(int marks) {
893 this.marks = marks;
894 }
895 }
896 -----student.hbm.xml-----
897 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
898 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
899 <!-- student.hbm.xml -->
900 <hibernate-mapping>
901 <class name="com.sathya.spring.model.Student" table="student">
902 <id name="studentId" column="sid"/>
903 <property name="studentName" column="sname"/>
904 <property name="marks" column="marks"/>
905 </class>
906 </hibernate-mapping>
907 -----applicationContext.xml-----
908 <beans xmlns="http://www.springframework.org/schema/beans"
909 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
910 xsi:schemaLocation="http://www.springframework.org/schema/beans
911 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
912 <!-- business class configuration -->
913 <bean id="db" class="com.sathya.spring.model.DemoBean">
914 <property name="curdDAO" ref="dao"/>
915 </bean>
916 <!-- dao class configuration -->
917 <bean id="dao" class="com.sathya.spring.dao.CurdDAOImpl"/>
918 </beans>
919 -----hibernate.cfg.xml-----
920 <!-- hibernate.cfg.xml -->
921 <!DOCTYPE hibernate-configuration PUBLIC
922 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
923 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
924
```

```
925 <hibernate-configuration>
926 <session-factory>
927 <!-- connection properties -->
928 <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver
929 </property>
930 <property name="hibernate.connection.url">
931 jdbc:oracle:thin:@localhost:1521:ORCL3</property>
932 <property name="hibernate.connection.username"> system</property>
933 <property name="hibernate.connection.password"> tiger</property>
934
935 <!-- hibernate properties -->
936 <property name="hibernate.dialect"> org.hibernate.dialect.Oracle10gDialect
937 </property>
938 <property name="hibernate.show_sql">true </property>
939 <property name="hibernate.hbm2ddl.auto"> update </property>
940
941 <!-- mapping resources -->
942 <mapping resource="com/sathya/spring/config/student.hbm.xml"/>
943 </session-factory>
944 </hibernate-configuration>
945 -----CurdDAO.java-----
946 //CurdDAO.java
947 //dao interface
948 package com.sathya.spring.dao;
949 import java.util.List;
950 public interface CurdDAO
951 {
952 void save(Object o);
953 void update(Object o);
954 List findStudents();
955 }
956 -----CurdDAOImpl.java-----
957 //CurdDAOImpl.java
958 //dao class
959 package com.sathya.spring.dao;
960 import java.util.List;
961 import org.hibernate.Query;
962 import org.hibernate.Session;
963 import org.hibernate.SessionFactory;
964 import org.hibernate.Transaction;
965 import org.hibernate.cfg.Configuration;
966 public class CurdDAOImpl implements CurdDAO
967 {
968 private static SessionFactory factory=null;
969 static
970 {
971 factory=new Configuration().
972 configure("com/sathya/spring/config/hibernate.cfg.xml").
973 buildSessionFactory();
974 }
975 @Override
976 public List findStudents() {
977 Session session=factory.openSession();
978 Query qry=session.createQuery("from Student s");
979 List list = qry.list();
980 return list;
981 }
982 @Override
983 public void save(Object o) {
984 Session session = factory.openSession();
985 Transaction tx=session.beginTransaction();
986 session.save(o);
987 tx.commit();
988 session.close();
989 }
990 @Override
```

```
991 public void update(Object o) {
992 Session session = factory.openSession();
993 Transaction tx=session.beginTransaction();
994 session.update(o);
995 tx.commit();
996 session.close();
997 }
998 }
999 -----Main.java-----
1000 //Main.java
1001 import org.springframework.context.ApplicationContext;
1002 import org.springframework.context.support.ClassPathXmlApplicationContext;
1003 import com.sathya.spring.model.DemoBean;
1004 public class Main
1005 {
1006 public static void main(String[] args)
1007 {
1008 ApplicationContext ctx=new ClassPathXmlApplicationContext(
1009 "com/sathya/spring/config/applicationContext.xml");
1010 Object o = ctx.getBean("db");
1011 DemoBean db=(DemoBean)o;
1012 db.insertStudent(111, "AAA", 600);
1013 db.updateStudent(111,"BBB", 900);
1014 db.selectStudents();
1015 }
1016 }
1017
```

```
1 -----Declarative Transaction -----
2 //Demo.java
3 package com.sathya.spring.transaction;
4 public interface Demo {
5 void method1();
6 }
7 //DemoBean.java
8 package com.sathya.spring.transaction;
9 import org.springframework.jdbc.core.JdbcTemplate;
10 public class DemoBean implements Demo
11 {
12 private JdbcTemplate jt;
13 public void setJt(JdbcTemplate jt)
14 {
15 this.jt=jt;
16 }
17 public void method1()
18 {
19 jt.update("insert into dept values(60,'EDU','HYD')");
20 jt.update("update emp set sal=6000 where empno=7788");
21 try {
22 Thread.sleep(10000);
23 } catch (InterruptedException e) {
24 e.printStackTrace();
25 }
26 jt.update("delete from sathya where stuid=111");
27 }
28 }
29 -----applicationContext.xml-----
30 <beans xmlns="http://www.springframework.org/schema/beans"
31 xmlns:aop="http://www.springframework.org/schema/aop"
32 xmlns:tx="http://www.springframework.org/schema/tx"
33 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
34 xsi:schemaLocation="http://www.springframework.org/schema/beans
35 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
36 http://www.springframework.org/schema/aop
37 http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
38 http://www.springframework.org/schema/tx
39 http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">
40 <!-- business bean configuration -->
41 <bean id="db" class="com.sathya.spring.transaction.DemoBean">
42 <property name="jt" ref="jt"/>
43 </bean>
44 <!-- transaction manager -->
45 <bean id="txm"
46 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
47 <property name="dataSource" ref="ds"/>
48 </bean>
49 <!-- JdbcTemplate -->
50 <bean id="jt"
51 class="org.springframework.jdbc.core.JdbcTemplate">
52 <property name="dataSource" ref="ds"/>
53 </bean>
54 <!-- DriverManagerDataSource -->
55 <bean id="ds"
56 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
57 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
58 <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
59 <property name="username" value="system"/>
60 <property name="password" value="root"/>
61 </bean>
62 <!-- transaction configuration -->
63 <tx:advice id="advice1" transaction-manager="txm">
64 <tx:attributes>
65 <tx:method name="method1"
66 propagation="REQUIRES_NEW"
```

```
67 isolation="READ_COMMITTED"
68 timeout="18"/>
69 </tx:attributes>
70 </tx:advice>
71 <aop:config>
72 <aop:pointcut id="pointcut1"
73 expression="execution(* com.sathya.spring.transaction.*.*(..))"/>
74 <aop:advisor advice-ref="advisor1"
75 pointcut-ref="pointcut1"/>
76 </aop:config>
77 </beans>
78 -----Main.java-----
79 import org.springframework.context.ApplicationContext;
80 import org.springframework.context.support.ClassPathXmlApplicationContext;
81 import com.sathya.spring.transaction.Demo;
82 public class Main
83 {
84 public static void main(String[] args)
85 {
86 ApplicationContext ctx = new ClassPathXmlApplicationContext
87 ("applicationContext.xml");
88 Demo db=(Demo)ctx.getBean("db");
89 try{
90 db.method1();
91 }catch(Exception e)
92 {
93 System.out.println(e);
94 }
95 }
96 }
97 =====Annotation Transaction=====
98 //Demo.java
99 package com.sathya.spring.transaction;
100 public interface Demo {
101 void method1();
102 }
103 //DemoBean.java
104 package com.sathya.spring.transaction;
105 import org.springframework.jdbc.core.JdbcTemplate;
106 import org.springframework.transaction.annotation.Isolation;
107 import org.springframework.transaction.annotation.Propagation;
108 import org.springframework.transaction.annotation.Transactional;
109 public class DemoBean implements Demo
110 {
111 private JdbcTemplate jt;
112 public void setJt(JdbcTemplate jt)
113 {
114 this.jt=jt;
115 }
116
117 @Transactional(propagation=Propagation.REQUIRES_NEW,
118 isolation=Isolation.SERIALIZABLE, timeout=5)
119 public void method1()
120 {
121 jt.update("insert into dept values(20,'RESEARCH','HYD')");
122 jt.update("update emp set sal=9900 where empno=7150");
123 try {
124 Thread.sleep(4000);
125 } catch (InterruptedException e) {
126 e.printStackTrace();
127 }
128 jt.update("delete from sathya where stuid=111");
129 }
130 }
131 -----applicationContext.xml-----
132 <beans xmlns="http://www.springframework.org/schema/beans"
```

```
133
134 xmlns:tx="http://www.springframework.org/schema/tx"
135 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
136 xsi:schemaLocation="http://www.springframework.org/schema/beans
137 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
138 http://www.springframework.org/schema/tx
139 http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">
140
141 <!-- business bean configuration -->
142 <bean id="db" class="com.sathya.spring.transaction.DemoBean">
143 <property name="jt" ref="jt"/>
144 </bean>
145 <!-- transaction manager -->
146 <bean id="txm"
147 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
148 <property name="dataSource" ref="ds"/>
149 </bean>
150 <!-- JdbcTemplate -->
151 <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
152 <property name="dataSource" ref="ds"/>
153 </bean>
154 <!-- DriverManagerDataSource -->
155 <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
156 <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
157 <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
158 <property name="username" value="system"/>
159 <property name="password" value="root"/>
160 </bean>
161 <tx:annotation-driven transaction-manager="txm"/>
162 </beans>
163 -----Main.java-----
164 import org.springframework.context.ApplicationContext;
165 import org.springframework.context.support.ClassPathXmlApplicationContext;
166 import com.sathya.spring.transaction.Demo;
167 public class Main {
168 public static void main(String[] args) {
169 ApplicationContext ctx = new ClassPathXmlApplicationContext(
170 ("applicationContext.xml"));
171 Demo db=(Demo)ctx.getBean("db");
172 db.method1();
173 }
174 }
175 =====Global Transaction=====
176 //AccountInter.java
177 package com.sathya.spring.global;
178 public interface AccountInter {
179 void transferMoney(int accno1,int accno2,int amount) throws Exception;
180 }
181 //AccountImpl.java
182 package com.sathya.spring.global;
183 import org.springframework.jdbc.core.JdbcTemplate;
184 import org.springframework.transaction.annotation.Transactional;
185 public class AccountImpl implements AccountInter
186 {
187 private JdbcTemplate jt1;
188 private JdbcTemplate jt2;
189 public void setJt1(JdbcTemplate jt1)
190 {
191 this.jt1=jt1;
192 }
193 public void setJt2(JdbcTemplate jt2)
194 {
195 this.jt2=jt2;
196 }
197 @SuppressWarnings("deprecation")
198 @Override
```

```
199 @Transactional(timeout=8)
200 public void transferMoney(int accno1, int accno2, int amount) throws Exception
201 {
202
203 int s1=jt1.queryForInt("select bal from Account1 where accno=?",accno1);
204 //deducting amount from balance
205 int s2 = s1-amount;
206 if(s2<500)
207 {
208 throw new Exception();
209 }
210 int s3=jt2.queryForInt("select bal from Account2 where accno=?",accno2);
211 //adding amount to balance
212 int s4 = s3+amount;
213
214 jt1.update("update Account1 set bal=? where accno=?",s2,accno1);
215 Thread.sleep(6000);
216 jt2.update("update Account2 set bal=? where accno=?",s4,accno2);
217 System.out.println("Transaction successful.....");
218 }
219 }
220 -----applicationContext.xml-----
221 <beans xmlns="http://www.springframework.org/schema/beans"
222
223 xmlns:tx="http://www.springframework.org/schema/tx"
224 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
225 xsi:schemaLocation="http://www.springframework.org/schema/beans
226 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
227 http://www.springframework.org/schema/tx
228 http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">
229 <!-- business class configuration -->
230 <bean id="accimpl" class="com.sathya.spring.global.AccountImpl">
231 <property name="jt1" ref="jt1"/>
232 <property name="jt2" ref="jt2"/>
233 </bean>
234 <!-- JdbcTemplate -->
235 <bean id="jt1" class="org.springframework.jdbc.core.JdbcTemplate">
236 <property name="dataSource" ref="ds1"/>
237 </bean>
238 <bean id="jt2" class="org.springframework.jdbc.core.JdbcTemplate">
239 <property name="dataSource" ref="ds2"/>
240 </bean>
241 <!-- DataSource configuration -->
242 <bean id="ds1" class="com.atomikos.jdbc.AtomikosDataSourceBean"
243 init-method="init" destroy-method="close">
244 <property name="uniqueResourceName" value="A"/>
245 <property name="xaDataSourceClassName"
246 value="oracle.jdbc.xa.client.OracleXADataSource"/>
247 <property name="xaProperties">
248 <props>
249 <prop key="databaseName">xe</prop>
250 <prop key="user">system</prop>
251 <prop key="password">root</prop>
252 <prop key="URL">jdbc:oracle:thin:@localhost:1521:xe</prop>
253 </props>
254 </property>
255 <property name="poolSize" value="1"/>
256 </bean>
257 <bean id="ds2" class="com.atomikos.jdbc.AtomikosDataSourceBean"
258 init-method="init" destroy-method="close">
259 <property name="uniqueResourceName" value="B"/>
260 <property name="xaDataSourceClassName"
261 value="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"/>
262 <property name="xaProperties">
263 <props>
264 <prop key="databaseName">test</prop>
```

```
265 <prop key="user">root</prop>
266 <prop key="password">root</prop>
267 <prop key="URL">jdbc:mysql://localhost:3306/test</prop>
268 </props>
269 </property>
270 <property name="poolSize" value="1"/>
271 </bean>
272 <bean id="atomikostm" class="com.atomikos.icatch.jta.UserTransactionManager"
273 init-method="init" destroy-method="close"/>
274 <bean id="txm" class="org.springframework.transaction.jta.JtaTransactionManager">
275 <property name="transactionManager">
276 <ref bean="atomikostm"/>
277 </property>
278 </bean>
279 <tx:annotation-driven transaction-manager="txm"/>
280 </beans>
281 -----Main.java-----
282 import org.springframework.context.ApplicationContext;
283 import org.springframework.context.support.ClassPathXmlApplicationContext;
284 import com.sathya.spring.global.AccountInter;
285 public class Main {
286 public static void main(String args[])
287 {
288 ApplicationContext ctx=new ClassPathXmlApplicationContext
289 ("applicationContext.xml");
290 Object o = ctx.getBean("accimpl");
291 AccountInter inter=(AccountInter)o;
292 try
293 {
294 inter.transferMoney(101,102,200);
295 }
296 catch(Exception e)
297 {
298 System.out.println("U r transaction unsuccessful");
299 }
300 }
301 }
302 ======Mail Test1=====
303 //TestBean.java
304 import org.springframework.mail.*;
305 import org.springframework.mail.javamail.*;
306 public class TestBean
307 {
308 private JavaMailSenderImpl mailSender;
309 public void setMailSender(JavaMailSenderImpl mailSender)
310 {
311 this.mailSender=mailSender;
312 }
313 public void sendEmail()
314 {
315 try
316 {
317 SimpleMailMessage smm =new SimpleMailMessage();
318 smm.setFrom("aabcd2011@gmail.com");
319 smm.setTo("abcd@hotmail.com");
320 smm.setSubject("Mail From Spring Application");
321 smm.setText("This is a sample mail from Spring Mail Application");
322 mailSender.send(smm);
323 System.out.println("Mail sent successfully");
324 }
325 catch(Exception e)
326 {
327 System.out.println(e);
328 }
329 }
330 }
```

```
331 =====spring-config.xml=====
332 <?xml version="1.0" encoding="UTF-8"?>
333 <beans xmlns="http://www.springframework.org/schema/beans"
334 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
335 xsi:schemaLocation="http://www.springframework.org/schema/beans
336 http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
337 <!-- spring-config.xml -->
338 <bean id="id1" class="org.springframework.mail.javamail.JavaMailSenderImpl">
339 <property name="host" value="smtp.gmail.com"/>
340 <property name="username" value="aabecd2011@gmail.com"/>
341 <property name="password" value="sathyatechnologies"/>
342 <property name="javaMailProperties">
343 <props>
344 <prop key="mail.transport.protocol">smtp</prop>
345 <prop key="mail.smtp.auth">true</prop>
346 <prop key="mail.smtp.port">465</prop>
347 <prop key="mail.smtp.socketFactory.port">465</prop>
348 <prop key="mail.smtp.socketFactory.class">javax.net.ssl.SSLSocketFactory</prop>
349 <prop key="mail.smtp.socketFactory.fallback">false</prop>
350 <prop key="mail.smtp.quitwait">false</prop>
351 <prop key="mail.smtp.starttls.enable">true</prop>
352 </props>
353 </property>
354 </bean>
355 <bean id="id2" class="TestBean">
356 <property name="mailSender" ref="id1"/>
357 </bean>
358 </beans>
359 -----Client.java-----
360 //Client.java
361 import org.springframework.context.*;
362 import org.springframework.context.support.*;
363 public class Client
364 {
365 public static void main(String args[])
366 {
367 ApplicationContext context =new ClassPathXmlApplicationContext
368 ("spring-config.xml");
369 Object o = context.getBean("id2");
370 TestBean tb=(TestBean)o;
371 tb.sendEmail();
372 }
373 }
374 =====JMS Application=====
375 -----sender-----
376 //MyMessageCreator.java
377 import javax.jms.*;
378 import org.springframework.jms.core.*;
379 public class MyMessageCreator implements MessageCreator
380 {
381 @Override
382 public Message createMessage(Session session)
383 {
384 TextMessage message = null;
385 try
386 {
387 message = session.createTextMessage();
388 message.setText("Hello World");
389 }
390 catch (JMSException e)
391 {
392 e.printStackTrace();
393 }
394 return message;
395 }
396 }
```

```

397 //MessageSender.java
398 import org.springframework.jms.core.*;
399 public class MessageSender {
400 private JmsTemplate jmsTemplate;
401 public void setJmsTemplate(JmsTemplate jmsTemplate)
402 {
403 this.jmsTemplate = jmsTemplate;
404 }
405 public void sendMessage()
406 {
407 MyMessageCreator creator=new MyMessageCreator();
408 jmsTemplate.send(creator); → send() method internally createMessage() of Message
409 }
410 }
411 -----spring.xml-----
412 <beans xmlns="http://www.springframework.org/schema/beans"
413 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
414 xsi:schemaLocation="http://www.springframework.org/schema/beans
415 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
416 <bean id="cf" class="org.springframework.jndi.JndiObjectFactoryBean">
417 <property name="jndiName" value="qcfactory"/>
418 <property name="jndiEnvironment">
419 <props>
420 <prop key="java.naming.factory.initial"> → JNDI properties [server to server it will change
421 com.sun.enterprise.naming.impl.SerialInitContextFactory</prop>
422 <prop key="java.naming.provider.url">iip://localhost:4848</prop>
423 </props>
424 </property>
425 </bean>
426 <bean id="q" class="org.springframework.jndi.JndiObjectFactoryBean">
427 <property name="jndiName" value="que"/>
428 <property name="jndiEnvironment">
429 <props>
430 <prop key="java.naming.factory.initial">
431 com.sun.enterprise.naming.impl.SerialInitContextFactory</prop>
432 <prop key="java.naming.provider.url">iip://localhost:4848</prop>
433 </props>
434 </property>
435 </bean>
436 <bean id="jt" class="org.springframework.jms.core.JmsTemplate">
437 <property name="connectionFactory" ref="cf"/>
438 <property name="defaultDestination" ref="q"/>
439 </bean>
440 <bean id="ms" class="MessageSender">
441 <property name="jmsTemplate" ref="jt"/>
442 </bean>
443 </beans>
444 -----Client.java-----
445 //Client.java
446 import org.springframework.context.ApplicationContext;
447 import org.springframework.context.support.ClassPathXmlApplicationContext;
448 public class Client
449 {
450 public static void main(String args[])
451 {
452 ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
453 MessageSender ms =(MessageSender)ctx.getBean("ms");
454 ms.sendMessage();
455 System.out.println("done");
456 }
457 }
458 -----receiver-----
459 //MessageReceiver.java
460 import javax.jms.*;
461 import org.springframework.jms.core.*;
462 public class MessageReceiver {

```

```
463 private JmsTemplate jmsTemplate;
464 public void setJmsTemplate(JmsTemplate jmsTemplate)
465 {
466 this.jmsTemplate = jmsTemplate;
467 }
468 public void receiveMessage()
469 {
470 Message message = jmsTemplate.receive();
471 TextMessage textMessage = null;
472 if (message instanceof TextMessage)
473 {
474 textMessage = (TextMessage)message;
475 try
476 {
477 System.out.println(textMessage.getText());
478 }
479 catch (JMSEException e)
480 {
481 e.printStackTrace();
482 }
483 }
484 }
485 }
486 -----spring.xml-----
487 <beans xmlns="http://www.springframework.org/schema/beans"
488 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
489 xsi:schemaLocation="http://www.springframework.org/schema/beans
490 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
491 <bean id="cf" class="org.springframework.jndi.JndiObjectFactoryBean">
492 <property name="jndiName" value="qcfactory"/>
493 <property name="jndiEnvironment">
494 <props>
495 <prop key="java.naming.factory.initial">
496 com.sun.enterprise.naming.impl.SerialInitContextFactory</prop>
497 <prop key="java.naming.provider.url">iop://localhost:4848</prop>
498 </props>
499 </property>
500 </bean>
501 <bean id="q" class="org.springframework.jndi.JndiObjectFactoryBean">
502 <property name="jndiName" value="que"/>
503 <property name="jndiEnvironment">
504 <props>
505 <prop key="java.naming.factory.initial">
506 com.sun.enterprise.naming.impl.SerialInitContextFactory</prop>
507 <prop key="java.naming.provider.url">iop://localhost:4848</prop>
508 </props>
509 </property>
510 </bean>
511 <bean id="jt" class="org.springframework.jms.core.JmsTemplate">
512 <property name="connectionFactory" ref="cf"/>
513 <property name="defaultDestination" ref="q"/>
514 </bean>
515 <bean id="mr" class="MessageReceiver">
516 <property name="jmsTemplate" ref="jt"/>
517 </bean>
518 </beans>
519 -----Client.java-----
520 import org.springframework.context.ApplicationContext;
521 import org.springframework.context.support.ClassPathXmlApplicationContext;
522 public class Client
523 {
524 public static void main(String args[])
525 {
526 ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
527 MessageReceiver mr =(MessageReceiver)ctx.getBean("mr");
528 mr.receiveMessage();
```

```
529 System.out.println("done");
530 }
531 }
532 ======Spring RMI application=====
533 -----Server-----
534 //DemoInter.java
535 public interface DemoInter
536 {
537 String sayHello(String uname);
538 }
539 //DemoImpl.java
540 public class DemoImpl implements DemoInter
541 {
542 public String sayHello(String uname)
543 {
544 return "Hello "+uname;
545 }
546 };
547 -----applicationContext.xml-----
548 <!-- applicationContext.xml -->
549 <beans xmlns="http://www.springframework.org/schema/beans"
550 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
551 xsi:schemaLocation="http://www.springframework.org/schema/beans
552 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
553 <bean id="di" class="DemoImpl"/>
554 <bean id="serviceExporter"
555 class="org.springframework.remoting.rmi.RmiServiceExporter">
556 <property name="service" ref="di"/>
557 <property name="serviceName" value="one"/>
558 <property name="serviceInterface" value="DemoInter"/>
559 </bean>
560 </beans>
561 -----Main.java-----
562 //Main.java
563 import org.springframework.context.ApplicationContext;
564 import org.springframework.context.support.ClassPathXmlApplicationContext;
565 class Main
566 {
567 public static void main(String[] args)
568 {
569 ApplicationContext ctx=new ClassPathXmlApplicationContext
570 ("applicationContext.xml");
571 }
572 }
573 -----Client-----
574 //TestBean.java
575 public class TestBean
576 {
577 private DemoInter demoInter;
578 //setter injection
579 public void setDemoInter(DemoInter demoInter)
580 {
581 this.demoInter=demoInter;
582 }
583 public void businessMethod()
584 {
585 try
586 {
587 String msg = demoInter.sayHello("RAM");
588 System.out.println(msg);
589 }
590 catch(Exception e)
591 {
592 e.printStackTrace();
593 }
594 }

```

```
595 }
596 //DemoInter.java
597 public interface DemoInter
598 {
599 String sayHello(String uname);
600 }
-----applicationContext.xml-----
601 <!-- applicationContext.xml -->
602 <beans xmlns="http://www.springframework.org/schema/beans"
603 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
604 xsi:schemaLocation="http://www.springframework.org/schema/beans
605 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
606 <bean id="tb" class="TestBean">
607 <property name="demoInter" ref="rpfb"/>
608 </bean>
609 <bean id="rpfb" class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
610 <property name="serviceUrl" value="rmi://localhost:1099/one"/>
611 <property name="serviceInterface" value="DemoInter"/>
612 </bean>
613 </beans>
614 -----
615 -----Main.java-----
616 //Main.java
617 import org.springframework.context.ApplicationContext;
618 import org.springframework.context.support.ClassPathXmlApplicationContext;
619 class Main
620 {
621 public static void main(String[] args)
622 {
623 ApplicationContext ctx=new ClassPathXmlApplicationContext
624 ("applicationContext.xml");
625 TestBean testBean = (TestBean)ctx.getBean("tb");
626 testBean.businessMethod();
627 }
628 }
629 ======Spring HTTP Invoker Service Application=====
630 -----HttpInvokerService-----
631 //DemoInter.java
632 public interface DemoInter
633 {
634 String sayHello(String uname);
635 }
636 //DemoImpl.java
637 public class DemoImpl implements DemoInter
638 {
639 public String sayHello(String uname)
640 {
641 return "Hello "+uname;
642 }
643 };
644 -----
645 <web-app>
646 <servlet>
647 <servlet-name>spring</servlet-name>
648 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
649 <load-on-startup>1</load-on-startup>
650 </servlet>
651 <servlet-mapping>
652 <servlet-name>spring</servlet-name>
653 <url-pattern>*.html</url-pattern>
654 </servlet-mapping>
655 </web-app>
656 -----
657 <!-- spring-servlet.xml -->
658 <beans xmlns="http://www.springframework.org/schema/beans"
659 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
660 xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```

661 >
662 <bean id="simpleUrlMappings"
663 class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
664 <property name="mappings">
665 <props>
666 <prop key="/test.html">serviceExporter</prop>
667 </props>
668 </property>
669 </bean>
670 <bean id="di" class="DemoImpl"/>
671 <bean id="serviceExporter"
672 class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
673 <property name="service" ref="di"/>
674 <property name="serviceInterface" value="DemoInter"/>
675 </bean>
676 </beans>
677 -----HttpInvokerClient-----
678 //TestBean.java
679 public class TestBean
680 {
681 private DemoInter demoInter;
682 //setter injection
683 public void setDemoInter(DemoInter demoInter)
684 {
685 this.demoInter=demoInter;
686 }
687 public void businessMethod()
688 {
689 try
690 {
691 String msg = demoInter.sayHello("Sathya");
692 System.out.println(msg);
693 }
694 catch(Exception e)
695 {
696 e.printStackTrace();
697 }
698 }
699 }
700 //DemoInter.java
701 public interface DemoInter
702 {
703 String sayHello(String uname);
704 }
705 -----applicationContext.xml-----
706 <!-- applicationContext.xml -->
707 <beans xmlns="http://www.springframework.org/schema/beans"
708 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
709 xsi:schemaLocation="http://www.springframework.org/schema/beans
710 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
711 <bean id="tb" class="TestBean">
712 <property name="demoInter" ref="hipfb"/>
713 </bean>
714 <bean id="hipfb"
715 class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
716 <property name="serviceUrl" value="http://localhost:2015/Service/test.html"/>
717 <property name="serviceInterface" value="DemoInter"/>
718 </bean>
719 </beans>
720 -----Main.java-----
721 //Main.java
722 import org.springframework.context.ApplicationContext;
723 import org.springframework.context.support.ClassPathXmlApplicationContext;
724 class Main
725 {
726 public static void main(String[] args)

```

↑  
 port  
 → HttpInvokerService [project name]

```
727 {
728 ApplicationContext ctx=new ClassPathXmlApplicationContext
729 ("applicationContext.xml");
730 TestBean testBean = (TestBean)ctx.getBean("tb");
731 testBean.businessMethod();
732 }
733 }
734 ======Spring Security App1=====
735 <!-- index.jsp -->
736 <center>
737 <h1>
738 Any one Can visit This Page
739 </h1>
740 <p>AdminPage</p>
741 <p>FacultyPage</p>
742 </center>
743 <!-- Admin.jsp -->
744 <center>
745 Welcome To Admin Page

746 Today Date is : <%=new java.util.Date()%>
747 <p>Logout
748 </center>
749 <!-- Faculty.jsp -->
750 Faculty Page visited.
751 <p>Logout
752 <!-- logout.jsp -->
753 <center>
754 You have successfully LoggedOut

755 start again
756 </center>
757 -----web.xml-----
758 <web-app>
759 <listener>
760 <listener-class>org.springframework.web.context.ContextLoaderListener
761 </listener-class>
762 </listener>
763 <context-param>
764 <param-name>contextConfigLocation</param-name>
765 <param-value>/WEB-INF/security-config.xml</param-value>
766 </context-param>
767 <welcome-file-list>
768 <welcome-file>index.jsp</welcome-file>
769 </welcome-file-list>
770 <filter>
771 <filter-name>springSecurityFilterChain</filter-name>
772 <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
773 </filter>
774 <filter-mapping>
775 <filter-name>springSecurityFilterChain</filter-name>
776 <url-pattern>/*</url-pattern>
777 </filter-mapping>
778 </web-app>
779 -----security-config.xml-----
780 <!-- security-config.xml -->
781 <beans xmlns:security="http://www.springframework.org/schema/security"
782 xmlns="http://www.springframework.org/schema/beans"
783 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
784 xsi:schemaLocation="http://www.springframework.org/schema/beans
785 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
786 http://www.springframework.org/schema/security
787 http://www.springframework.org/schema/security/spring-security-3.0.3.xsd">
788
789 <security:http use-expressions="true">
790 <security:intercept-url pattern="/index.jsp" access="permitAll" />
791 <security:intercept-url pattern="/secure/Admin.jsp" access="hasRole('admin')"/>
792 <security:intercept-url pattern="/secure/Faculty.jsp" access="hasRole('faculty')"/>
```

```
793 <security:http-basic/>
794 <security:logout logout-success-url="/logout.jsp"/>
795 </security:http>
796 <security:authentication-manager>
797 <security:authentication-provider>
798 <security:user-service>
799 <security:user name="satya" password="tech" authorities="admin" />
800 <security:user name="ssreddy" password="ssreddy" authorities="faculty" />
801 </security:user-service>
802 </security:authentication-provider>
803 </security:authentication-manager>
804 </beans>
805 ======Spring Security App2=====
806 <!-- index.jsp -->
807 <center>
808 Any one Can visit This Page
809 <p>Admin Page</p>
810 <p>Faculty Page</p>
811 </center>
812 <!-- login.jsp -->
813 <html>
814 <head>
815 <title>Security WebApp login page</title>
816 </head>
817 <body bgcolor="#cccccc">
818 <blockquote>
819 <h2>Please enter your user name and password:</h2>
820 <p>
821 <form method="POST" action="j_spring_security_check">
822 <table border=1>
823 <tr>
824 <td>Username:</td>
825 <td><input type="text" name="j_username"></td>
826 </tr>
827 <tr>
828 <td>Password:</td>
829 <td><input type="password" name="j_password"></td>
830 </tr>
831 <tr>
832 <td colspan=2 align=right><input type=submit
833 value="Submit"></td>
834 </tr>
835 </table>
836 </form>
837 </blockquote>
838 </body>
839 </html>
840
841 <!-- login_failed.jsp -->
842 <h1> U r Credentials are not valid </h1>
843

844 enter again
845
846 <!-- logout.jsp -->
847 <center>
848 You have successfully LoggedOut

849 start again
850 </center>
851
852 <!-- Faculty.jsp -->
853 Faculty Page visited.
854 <p>Logout
855
856 <!-- Admin.jsp -->
857 <center>
858 Welcome To Admin Page

```

```
859 Today Date is : <%=new java.util.Date()%>
860 <p>Logout
861 </center>
862 -----web.xml-----
863 <web-app>
864 <listener>
865 <listener-class>org.springframework.web.context.ContextLoaderListener
866 </listener-class>
867 </listener>
868 <context-param>
869 <param-name>contextConfigLocation</param-name>
870 <param-value>
871 /WEB-INF/security-config.xml,/WEB-INF/applicationContext.xml
872 </param-value>
873 </context-param>
874 <welcome-file-list>
875 <welcome-file>index.jsp</welcome-file>
876 </welcome-file-list>
877 <filter>
878 <filter-name>springSecurityFilterChain</filter-name>
879 <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
880 </filter>
881 <filter-mapping>
882 <filter-name>springSecurityFilterChain</filter-name>
883 <url-pattern>/*</url-pattern>
884 </filter-mapping>
885 </web-app>
886 -----security-config.xml-----
887 <beans xmlns:security="http://www.springframework.org/schema/security"
888 xmlns="http://www.springframework.org/schema/beans"
889 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
890 xsi:schemaLocation="http://www.springframework.org/schema/beans
891 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
892 http://www.springframework.org/schema/security
893 http://www.springframework.org/schema/security/spring-security-3.2.xsd">
894 <security:http use-expressions="true">
895 <security:intercept-url pattern="/index.jsp" access="permitAll"/>
896 <security:intercept-url pattern="/secure/Admin.jsp" access="hasRole('admin')"/>
897 <security:intercept-url pattern="/secure/Faculty.jsp" access="hasRole('faculty')"/>
898 <security:form-login login-page="/login.jsp"
899 authentication-failure-url="/login_failed.jsp"/>
900 <security:logout logout-success-url="/logout.jsp"/>
901 </security:http>
902 <security:authentication-manager>
903 <security:authentication-provider>
904 <security:jdbc-user-service data-source-ref="ds"
905 users-by-username-query="select username,password,enabled from users where
906 username=?"
907 authorities-by-username-query=" select u.username, ur.authority from users u,
908 user_roles ur where u.user_id = ur.user_id and u.username =?"/>
909 </security:authentication-provider>
910 </security:authentication-manager>
911 </beans>
912 -----applicationContext.xml-----
913 <beans xmlns="http://www.springframework.org/schema/beans"
914 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
915 xsi:schemaLocation="http://www.springframework.org/schema/beans
916 http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">
917 <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
918 <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
919 <property name="url" value="jdbc:mysql://localhost:3306/test"/>
920 <property name="username" value="root"/>
921 <property name="password" value="root"/>
922 </bean>
923 </beans>
924
```