SOFTWARE TRAINING & DEVELOPMENT
11 YEARS
TRAINING EXCELLENCE

SPRING

COURSE MATERIAL

NARESH technologies™

Opp.Satyam Theatre, Ameerpet, Hyderabad - 16
Ph: 040-23746666, 23734842 Cell: 9000994007 / 08

.O. Kaba Plaza, 3rd Floor, LB Road, Adyar, Chennai - 20. Phone : 044-6555 5557, 6555 5558

info@nareshit.com www.nareshit.com nareshit nareshitech

Prativa Sahoo

Spring Index

Introduction

1) Introduction 01-03
2) Spring 1.X Modules, 2.x,3.x,4.x overview diagrams 04-07
3) MVC architecture 07-08
4) Dependency injection 09-11
5) POJI (Interface) 11-14

CoreModule:

1) Dependency Injection Introduction 15-23
2) DI(Application 2) 23-26
3) Stratgery DesingPattern 27-37
4) Resolving/identifying params in constructor injection 38-41
5) Difference between constructor and setter injection 42
6) Inner Beans 42-47
7) Bean Inheritance 48-50
8) Collection Merging 51-52
9) Bean aliasing 53-55
10) Factory Method 56-57
11) Singletone class, BeanScopes 58-62
12) Dependency-check 63-64
13) Nested BeanFactory 65-68
14) P-name space C-name space 68-70
15) (Applicaion Contaxt continar) Ioc continar 71-77
16) Working with properties files 77-78
17) In18 79-83
18) Event handling 84-88
19) Bean life cycle 83-91
20) xxxxAware Interfaces 92-94
21) FactoryBean 95-103
22) Method Replacement 104-107
23) LookupMethod injection 108-117
24) Beanlife cycle diagram 118-120
25) BeanFactory PostProcesser 121-127
26) Spring core module Annotations 128-137
27) Java config annotations 138-147

28) Properties file with annotations 147-149
29) 100% java code 149-152

Spring JDBC/DAO Module:

1) Introduction 153-156

2)Jdbc Templet 157-162

3)JNDI Registery 163-170

4)Call back Interfaces 171-177

5)Batch Processing/updating 178-183

6)Named parameter Jdbc Templet 184-189

7)Simple JdbcInsert, SimpleJdbcCall 190-197

8)Mapping sql operation with subclasses 198-207

AOP module:

1)AOP Introduction 208-210

2)AOP Principles 210-211

3)Declarative approach 211-229

4)Before advice 229-237

5)After advice 237-241

6)Throws advice 241-244

7)Programatic spring AOP appropch ,PointCut 245-251

8)AspectJAOP 252-262

9)Annotation driven AspectJAOP 264-269

10)Transaction Management 270-279

11)declarative Transaction Managment 280-283

12)Annotation driven TransactionMangment 283-286

13)100% java code 287-293

Spring MVC Module:

1) Introduction 294-299

2)Controller Classes 299-302

3)ViewResolver 302-303

4)Parameterizable View Controller 304-307

5)UrlFileNameViewController 308-309

6)Abstract Controller 310-321

7)Command Class,Abstract Command Controller 322-335

8)Internall flow of Command Controller 336

9)Simple Form Controller 336-346

10)Form Validation using SimpleFormController 346-355

11)internal flow of validation 356-370

12)ReferenceData(HttpRequest) 370-379

13)MultiActionController 379-383

14)AbstractWizardFormController 383-390

15)HandlerMapping 390-397

16)HandlerInterceptor 398-402

17)ViewResolver 402-407

18)I18n 408-412

19)Annotation driven Mvc 412-417

20)Annotation base abstractCommandController 418-423

21)Annotation base SimpleFormController 423-431

22)FileupLoading and downloading 432-439

23)MVC with Tiles 440-445

24)Handling Exceptions 445-448

25)100% java code based spring mvc 449-459

Spring Security:

1) Introduction 459-460

2)Example Application 461-463

Spring ORM: 464-475

Spring Boot 476-501

Spring FAQs 502-511

Additional questions 512-529

Introduction

->C,C++, Java are programming languages.
->JDBC, Servlet, EJB,JSP,JMS,java mail,jta,jpa and etc are java technologies.
->Struts,SPRINGFRAMEWORK,Spring,Hibernate and etc are Java Frameworks.

Programming Language:
It is directly installed software having features to develop software applications.
It defines syntaxes (Rules), semantics (Structures) of programming.
These are base to create software technologies, frameworks, tools, Database software, operating systems and etc..
These are raw materials of software programming.
eg: C,C++,C#,Java.

Software Technology:
It is software specification having set of rules and guidelines to develop software by using programming language support.
Software technology is not installable but software created based on software technology is installable. Working with this software is nothing but working with software technologies.
In Java environment, technology rules are nothing but interfaces and guidelines are nothing but classes. Abstract class represent both rules and guidelines.
eg: JDBC,JSP, Servlet and etc....
JDBC technology gives rules and guidelines to develop driver using java language. Working with JDBC Driver is nothing but working with Jdbc technologies.
There are 2 types of software technologies.

1.Open Technologies
2.Proprietary Technologies.

Open Technologies:
Here the rules and guidelines of technology are open to all software vendor comparies to develop software.
eg: All Java Technologies (JDBC, Servlet, JSP, etc..)

Proprietary Technologies.
Here the vendor company who has given technology is only allowed to develop software based on technologies.
eg: All Microsoft Technologies (Like ASP.NET)

Frameworks:
Framework is installable software that uses existing technologies internally to simplify the application development having ability to generate common logic of application internally.
Framework is installable software that provides abstraction layer on existing technologies to simplify the application development process.
eg: : Struts, Spring, Hibernate,...
While working with technologies the programmer has to develop both common logic and specific logics of application.
While working with frameworks the programmer has to develop only application specific logics because the common logics will be generated by the framework dynamically.

Every framework uses technology internally but it never makes programmer to know about it. This is nothing but providing abstraction layer (hiding implementations/internals)**Plain JDBC App:**

->Register JDBC Driver (load Driver class)
->Establish the Connection
->Create Statement Object
->Send and execute SQL query in Database software
->Gather results and process results.
->Close Jdbc Objects

Spring jdbc App:

->Get JdbcTemplate Object.
->Send and execute SQLQuery in Database software
->Gather results and process results.

While working with technology we need to write some common logics in every application of that technology. This is called ***boiler plate code*** problem.
While working with frameworks we just need to develop application specific logics that means it solves **boiler plate code** problem.
➔Based on kinds of application and logics we develop, there are 3 types of java Frameworks.

Web Application Frameworks:

Provides Abstraction layer on servlets, jsp technology
Simplifies MVC Architecture based web application development
JSF ->From Sun MS (Oracle)
Struts-> From Apache
Webwork->from open symphony.
ADF->from Oracle
Spring MVC ->From interface

ORM Frameworks:

Provides Abstraction layer on Jdbc technology.
Simplifies Object based O-R mapping persistence logic development.
Allows to portablity/Database independent persistence logic.
Hibernate -> from Softtraa (RedHat)
TOPLink ->Oracle Coorperation
OJB-> Apache
iBatis->from Apache
JDO->from adobe

Application Frameworks:

Provides abstraction layer on multiple technologies like JDBC, servlets, jsp, jndi,jms, jta, jaas,EJB,rmi and etc..
Allows to develop all kinds of logic like business logic, persistence logic, presentation logic, integration logic and etc.
Allows to develop all kinds of applications like standalone applications, web applications distributed applications and etc..
eg: Spring-> from interfaces
Based on mode of development they allow, there are 2 kinds of frameworks.

Invasive Framework:
Here the classes of application should extend from framework api class or should implement framework api interface.
This process makes api classes as tightly bounded classes of frameworks.
We cannot change framework of the application/ project by changing the libraries.

Non- Invasive Framework:
Here the class of application need not to extend from framework class or need not to implement framework api interfaces. So our application classes are not tightly bound with framework.
We can change framework of application/ project by changing the libraries (jar files)

eg: Struts 2.x,Spring, Hibernate, and etc..
Evaluation of spring
Applets-> For Gaming (1995)
JavaBean-> for B.logic/service logic development (1996)
(Disadvantage: Cann't apply services, cann't be accessed from remote clients)
Security, transactions, logging. Connection pooling and etc are called services/ middleware services.
EJB ->for Business logic and service logics development (1998)

Advantages:
Allows both local and remote clients
Allows to apply middleware services.

Disadvantages:
Heavy weight and complex.
EJB is suitable for banking projects.

Spring (2003)
Inititally for business logic development with ordinary classes allowing to apply middleware services.
Now spring can be used to develop all kinds of logics and applications in light weight environment.

Before Frameworks:
Programmer directly uses technologies to develop both common and specific logics of application.

With Frameworks:
Programmers uses frameworks to develop only specific logics because the common logics will generated by framework internally by using technologies support.

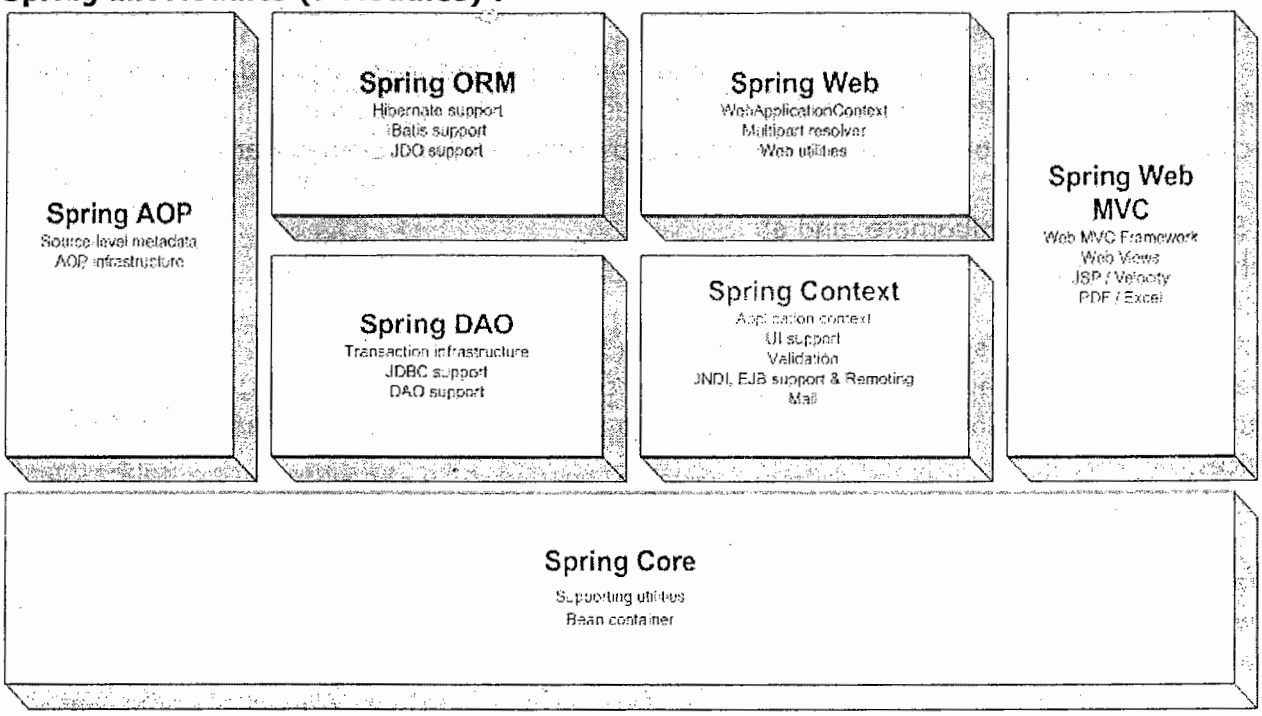
Is the spring replacement for struts?
Ans: Using struts we can develop only web application. There is no provision in struts to develop and use middleware services / aspects (security, logging and etc..)
Using spring we can develop all kinds of applications and all kinds of logics including web applications. It also provides environment to develop aspects/middleware services.
Spring is more powerful than struts

Is the Spring replacement for JEE/JSE technologies?

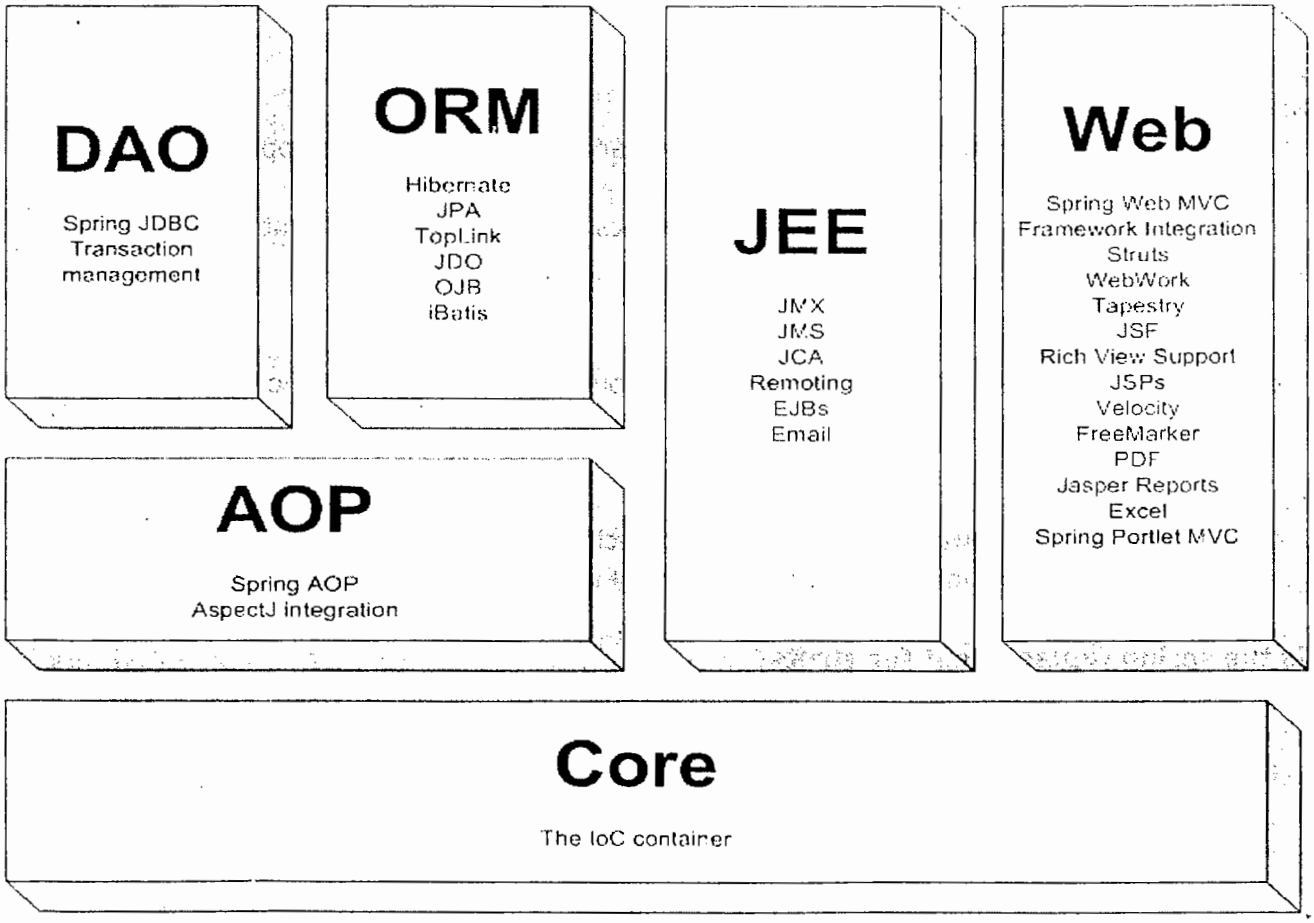
Ans : No

Spring complement JEE/JSE technologies by using them internally in the application development process.

Spring 1.X Modules (7 Modules) :



Spring 2.x Overview Diagram:



Spring 2.x web module : Spring 1.x web + spring 1.x web mvc module.
Spring 2.x JEE modules : Spring 1.x context module.5

Spring Core:
This module is base for other modules, provides containers that required to manage various spring components/resources.
Useful to develop standalone applications, Designed supporting dependency injection (injecting values to Objects dynamically)

Spring DAO/JDBC:
Provides abstraction layer on plain JDBC to simplify the JDBC style persistence logic.and
Exception
Re-Throwing concept is used to designed DAO Exception herirachy

Spring ORM:
Provides abstraction layer on multiple ORM framework (like Hibernate) and simplifies Object Based O-R mapping persistence logic.

Spring Web:
Provides plug-in to make spring applications communicatable from other web framework applications like Struts, JSF and etc ...applications.

Spring Web MVC:
Spring own web framework to develop MVC architecture based web applications with plan classes by using orbiter methods of the classes.

Spring Context:
Provides abstraction layer on multiple JSE, JEE module technologies to simplify the development of the comman logics.
Mailing applications
Distributes applications
Messaging applications
Scheduling applications
and etc.....

Spring AOP:
Provides a new methodology of programming to develop aspect/ middleware services logics and to apply them on spring applications.

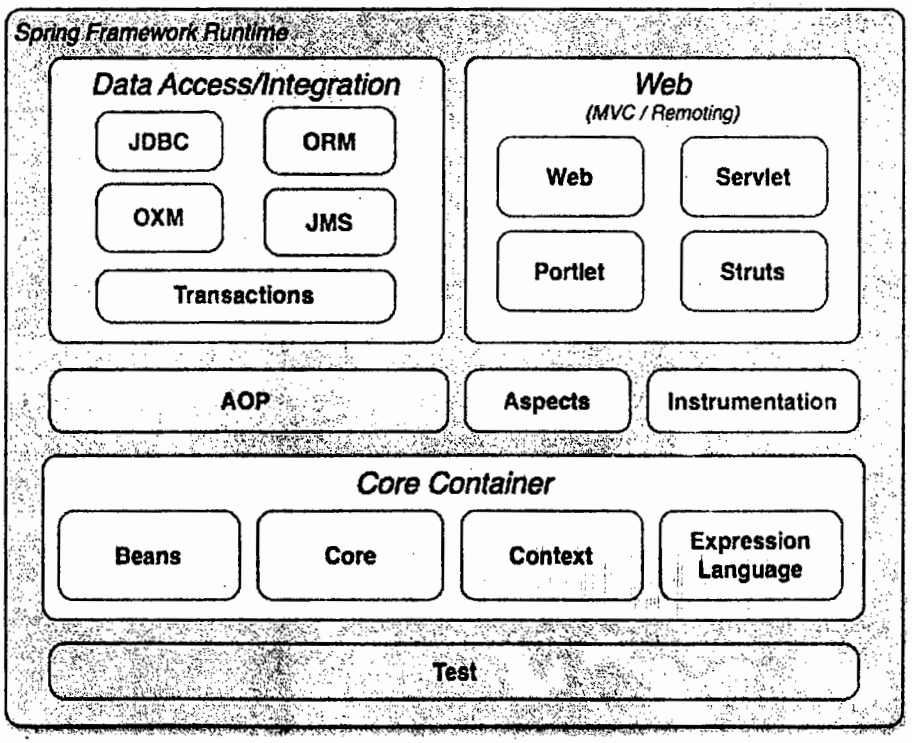
Before AOP:

```
class Bank {
public void withdraw (-,-){
  Security logic
  Logging logic
  Transaction logic
 withdraw logic (bal=bat-amt)
}}
public void deposit(){
Security logic
Logging logic
Transaction logic
Deposite logic (bal=bal+amt)
}}
```

With AOP:

```
class Bank{
 public void withdraw (-,-){
Withdraw logic(bal=bal-amt);
}
public void deposit(){
Deposit logic (bal=bal+amt);
}
}
```

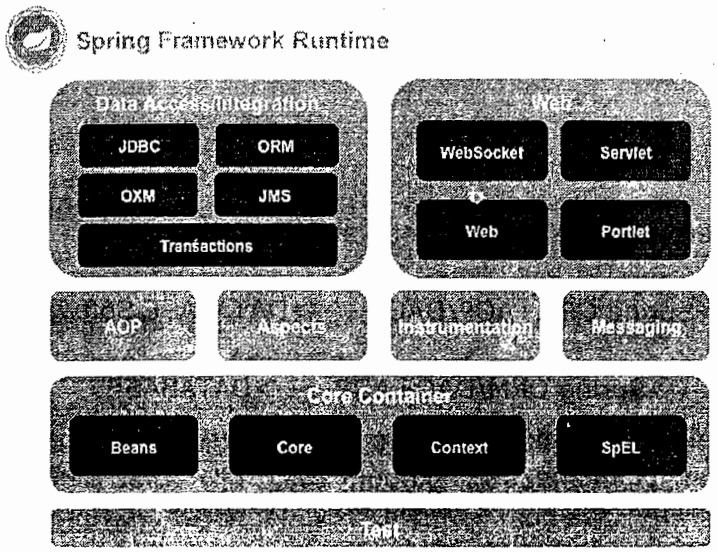
Spring 3.x Overview Diagram:



Spring 3.x is having 20 modules that are grouped into core container, AOP, instrumentation,

Test, DataAccess/integration, MVC/ remoting groups.
Test module allows to perform unit testing (programmers testing on his own piece of code) junit does not allow to mock objects (dummy objects), where spring 'Test' module allows us to work with mock object.
OXM (Object xml mapping) is given to convert java objects to xml tags and vice-versa (alternate to JAXB)
Spring 2.x JEE services they placed in DataAccess/integration, MVC/remoting groups.
Portlet supports allows us to develop "portal" applications multiple windows (web pages) in a webpage.

Spring 4.x override diagram:



Spring 3.x is compatible with jdk 1.5
Spring 4.x is compatible with jdk1.6 and support all jdk1.8 features. In this version all deprecated classes, methods are removed
In spring 4.x messaging a sub framework in spring that given alternate to *'jms'* to get messages based asynchronous communication between components.
Asynchronous communication allows the client to given next request or to client side operations without given request related response from server.
Web socket is a web lavel protocol that is designed on the top of http protocol for full 2-way communication.
Industry uses "**MVC" as defacto architecture** to develop java based medium/large scale web applications.
M-> Model ->Data + Business Logic+ persistence logic (like account officer)
V->View ->presentation logic (like Beautician)
C->Controller ->Integration logic (like Traffic police/supervisor)

Note:
Integration logic controls and monitors all the activities of application execution. It is responsibleto get communication between view layer and model layer components.
Jsp----------->servlets----------> java class/java Bean------>DATABASE software
(V) (C) (M)
Jsp----------->servlets----------> EJB Session Bean-------->Entity Bean------>DATABASE software
(V) (C) (M) (M)

Jsp------->servlets---------> EJB Session Bean-------->DAO class (JDBC)------->DATABASE software.
(V) (C) (M) (M)

Struts---------->EJB Session Bean--------->DA0 Class (JDBC)------->DATABASE Software.
(V) (C) (M)

DAO: Data Access Object:
The java class that separates persistence logic from another logics and makes the persistence logic as reusable logic and flexible logic to modify is called DAO.
Jdbc is light weight but does not allows to develop Object based persistence logic. EJB entity Beans heavy weight but allows to develop Object based persistence logic. To overcome both problems we can use "Hibernate"
Struts----->EJB session Bean--------> hibernate------>DATABASE software

Struts-------->Spring-Hibernate---------->DATABASE Software.

Struts--------->WebServices---------->Hibernate->DATABASE Software.

Struts---------->Spring JEE/Bean---------->Spring ORM+AOP/DAO------>DATABASE Software.

Struts--------->Spring JEE/Bean--------->Spring ORM/DAO+AOP----->DATABASE Software.

Struts---------->Spring WebServices/Web services-------->Spring ORM+AOP/DAO----->DATABASE Software.

Note:while developing mvc project we use spring module like core,aop for developing local service logic.like s logic of cricbuzz.com and we use web services to devlope external service logics likes icc score provides for cricbuzz, espncricinfo

Spring:
|----->**Type:** Java application framework
|-->**Version**:4.x (Compactable with jdk 1.6+)
3.x (Compactable with jdk 1.5+)
|-->**vendor :** interface 21
|-->**Creator:** Mr.Rod Jonshon
|-->Springsoftware comes as set of jars
!-->**website:** http://spring.io
!-->**To downloadsoftware:** Download as zip file from spring.io
spring-framework-4.2.2.RELEASE-dist.zip
Spring-framework-3.2.Release-with-docs.zip.
|->**To installsoftware:** Extract zip file.

Spring 4.1.6/3.2.6 software installation gives
libs->having jarfiles.
Docs-->apidocs, reference docs (PDF)
Schemas->xsd file having schema rules.
Every DTD and XSD file contains rules to construct xml documents.
DTD->document type definition
XSD->xml schema definition.

Spring Definition:

Spring is an **opensource, light weight, loosly coupled, aspect oriented, dependency injection based** java application framework to develop various java applications.
Along with spring software installation we get its source code. This makes spring as open source.
Open source software are not only free software. They also expose their source code along with installation.
Spring is lightweight because
->Spring framework comes as zip file having less size
->Spring gives 2 lightweight containers which can activated anywhere in the application.
To work with spring container we donot need any application server/webserver software.
We can use specific modules of spring without having any link with other modules.

eg: We can use spring core module alone or spring core and spring DAO module together without any link with other modules.
If degree of dependency is less between modules, then they are called loosly coupled components.
eg: Tv and remote.
If degree of dependency is more between modules then they are called tightly coupled components.
eg: Keyboard and system.

In spring framework core module is base module for all modules i.e we should use every module along with core module. But remaining modules of spring can be used alone without having any dependency with other modules. The degree of dependency is very less between other modules of spring.This makes spring as loosly coupled framework.

AOP is a new methodology of programming that compliments OOP to separate secondary middleware services from primary business logic and allows to link them at runtime dynamically through some configuration. This gives advantages of reusing middleware service logic and flexibility of enabling or disabling middleware services logic on primary logis (like security, transaction and etc....)

Dependency Lookup:
If the resource/ class/ Object is searching and getting is dependent values from other resources then it is called dependency lookup. Here the resource pulls the dependent values.
->The way we get Objects from Jndi registry is called "Dependency lookup"
->The way we get Jdbc connection Object from JDBC Connection pool is called "Dependency Lookup"
->If student is getting course material only after putting request for it is called dependency lookup. main component should write logic to get dependency

Dependency injection (Inversion of control/IOC):
If underlying server/container/ framework/runtime (like JRE) assigns values to resources/class/objects dynamically at run time then it is called ***dependency injection***.
->Here dependency values will be pushed to the resource.
->eg: JVM calling constructor automatically to assign initial values of object when object is created. Adva: dynamically assign dependent object to main object
Disadva: the underlaying container of server both neccery and unneccesary objects to main objects
->Servlet container injects ServletConfig object to our servlet class Object by calling

init(ServletConfig config) method when servlet Container creates our Servlet class object.
->Student gets course material the moment he register for course.

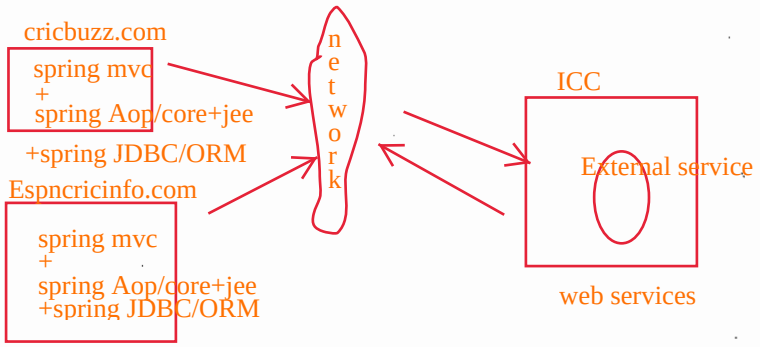
What is pojo class, POJI, Bean/Comonent class, Spring Bean?

POJO class (Plain old java Object class)

The java class without any specialities i.e it is an ordinary java class.
While developing this class we need not to follow any serious rule.
The java class that is not extending from technology or framework api class and not implementing technology/ framework api interfaces is called pojo class.

eg:

```
class Demo
{.......
.......
}
It is pojo class
class Test implements Serializable{
.............
.............
}
Pojo class
```



eg :

```
class Test extends Demo
{
.......
......
}
class Demo{
.............
.............
}
 eg:
class Test extends Thread{
.........
........
}
"Test","demo"->pojo class
```

List of view Layer Technologies
Html,xhtml,jsp,jquery,angularjs,DoJO,Zimbra etc
List of Controller Technologies
servlet ,ServletFilter
List of model layer technologies/Frameworks to develop B.logics
java class,java class+java Bean, Spring core+AOP,spring JEE,RMI,EJB
List of Model Layer Technologies/Frameworks to develop Persistence logic
jdbc,Entity Beans,Spring JDBC,Spring ORM,Hibernate,Ibatis,JDO etc
List of Frameworks to develop View and Controller layer logics
structs,spring MVC,jsf,webwork,Tapstrey,ADF etc
List Technologies/ Frameworks to develop external services
webservices(SOAP/Restfull),EJB,RMI,CORBA etc

eg :

```
class Test extends HttpServlet{
...........
.........
}
"Test" is non pojo class.
eg:
class Test implements java.sql.Connection{
............
.........
}
```

"Test" is non pojo

eg:

```
class Test {
public void m1(Statement st){
.................
.................
}
}
```

"Test" is pojo class.

eg:

```
class Test extends Demo{
................
.....................
}
class Demo implements java.rmi.Remote{
...............
...........
}
```

"Demo", "Test" are not pojo class

eg:

```
@Entity
class Test{
..........
.........
}
```

Test is a pojo class.

No need to include annotation and internal methods of tech or framework.

Don't consider annotations and api's used inside the methods of a class to check whether the class is pojo class or not.

POJI (Plain Old Java Interafce)

It indicates an interface without specialities i.e an ordinary interfaces.

The interface that is not extending from technology/ framework api interface is called poji.

eg:

```
interface Test{
......
.........          "Test" is pojo
}
```

eg:

```
interface Test extends Serializable{
...............
...................
}
```

"Test" is poji

```
interface Test extends java.rmi.Remote{
.................
.................
}
"Test" is not poji
```

eg:

```
interface Test extends Demo
{
..........
.........
}

interface Demo {
.......
.........
}
```

Test, Demo are pojis

The frameworks that support pojo/ poji model programming are called non invasive frameworks or light weight frameworks.

eg:
Hibernate, Spring, Struts 2.x etc

Java Bean:
It is a java class that is developed with some standards.The standards are
->Must be public class and should not be final or abstract class.
->Can be implement Serializable, if needed.
->Must have direct or indirect 0-parameter constructor.
->Member variables are called bean properties and they must be taken as private.
->Every bean property should have setter and getter methods (public)

eg:

```
public class StudentBean{
//bean properties
private int st,
private String name;
//setter and getter
...........
..............
}
```

->In real time javabean is used as helper class to represent data with multiple values in the form of object and to send those objects from one layer to another layer.
->JavaBean object is acceptable format to send and receive data across the multiple layers irrespective of technologies or that are used in the layer to develop the logics.
->The JavaBean class whose object holds input values given by client/enduser is called VO class.

->The JavaBan class whose object holds data needed for persistence operation (DAO Class) is called as (Business class) BO class
->The java bean whose data is shippable over the network is called DTO class
BO class: Business Object Class
VO class: Value Object class.
DTO class: DataTransferObject class.

Bean Class/Component Class:
It is a java class that contains member variables and methods having business logic manipulating the data of member variables.

eg:

```
class CardProcessor{
private int cardNo;
public int processcard (Int cardNo){
..............
.............      //business logic
............
}
public boolean blockcard (int cardNo)
{
..............
..............      //business logic
..............
}
}
```

The Bean/component class can extend or implement any class or interface.

Spring Bean:

- The java class whose object can be managed by spring container is called springBean.
- SpringBean can be pojo class, java bean class.
- Bean/component class, but its object must be created and managed by spring container.
- Spring Bean can be user defined class/pre-defined class /third party class.
- We can't take abstract class/interface as springBean because it can't create object for abstract class or interface.
- To make java class as spring Bean it must be configure in spring bean configuration file (xml) to make spring container to create and manage that spring Bean class object.

Important points:
Spring Bean can be pojo class? (Yes)
Every java bean is pojo class? (Yes)
.Every pojo class is java Bean? (No)
Every spring bean is java Bean? (No)
Spring Bean can be a java Bean? (Yes)
Spring Bean cannot implement spring api interface? (No)
Any component/bean class can be taken spring as Bean? (Yes)
Spring gives 2 light weight containers which can be started in any application by just creating object for container classes.
Any java class or application that can manage the life cycle of given resource (class) is called container.

Spring containers are :

1.**BeanFactory** (Basic container)
(performs spring bean management and dependency injection)

2.**ApplicationContext** (Advanced Container)
(internally uses BeanFactory container so can perform advanced operations along with Bean management and dependency injection)
To activate/ start BeanFactory container we should create object of a class that implements "BeanFactory".

eg:

Beanfactory Factory= new XmlBeanFactory (Resource object)

|→Points to spring bean config file (xml file)

Spring containers (light weight) are not replacement/ alternate for servlets, jsp containers (heavy weight).

Spring Core Module

Base module for other spring module
Gives two IOC containers/spring containers.
1)BeanFactory
2)ApplicationContext.
When it is used alone we can develop standard applications.
When it is used along other technologies we can develop model layer business logic by taking advantage of dependency injection.
Talks about spring bean life cycle management and dependency injection.

Dependency Injection:
If the underlying server or container dynamically assigns the dependent values to our resource (class/object) then it is called dependency injection. There are multiple ways of performing dependency injection
a)Setter injection (Container calls *setXxx(-)* to inject dependent values).
b)Constructor injection (container uses parameterized constructor to create bean class object and to inject value to that object)
c) interface injection (Aware injection)
d)Method injection
e)Lookup Method injection
and etc....

Setter injection sample code:

```
//WishGenerator.java (POJO class)
package com.nt.beans;
import java.util.Date;
public class WishGenerator {
    //bean properties
    private  String  name;
    private Date date;
    public WishGenerator(){
            System.out.println("WishGenerator:0-param constructor");
    }
```

//setter methods for setter injection

```
    public  void  setName(String name) {
            System.out.println("WishGenerator:setName(-)");
            this.name = name;
    }
    public void setDate(Date date) {
            System.out.println("WishGenerator:setDate(-)");
            this.date = date;
    }
    // B.method
    public String  generateWishMsg(){
            return "Good Morning:"+ name+"--->"+date;
    }
}//class
```

applicationContext.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.3.0.xsd">

<bean id="dt" class="java.util.Date"/>
<bean id="wsg"  class="com.nt.beans.WishGenerator" >
<property name="name"  value="raja"/>
<property name="date"  ref="dt"/>
</bean>
</beans>
```

Any *<file-name>.xml* can be taken as spring configuration file. But it is recommended to take "applicationContext.xml" in standalone spring applications.

In springBean configuration file we configure
a)classes as springBeans
b)Dependency injection configuration
c)Spring bean lifecycle Configuration and etc..

We must supply springBean configuration file while activating spring container.
Every spring Bean is identified with its bean id. *<property>* (tag) is used to configure bean property for **setter injection**.

In the above configurs spring container creates "Date", WishGenerator" class objects using 0-param constructor and calls *setName(-)* and *setDate (-)* methods to perform setter injection

Constructor injection Sample Code:

```
//WishGenerator.java
package con.nt.beans;
public class WishGenerator {
//bean properties
private String name;
private Date date;
//param constructor for constructor injection
public WishGenerator (String name, Date date){
this.name=name;
this.date=date;
//B.method
.........................
.........................
}
}
```

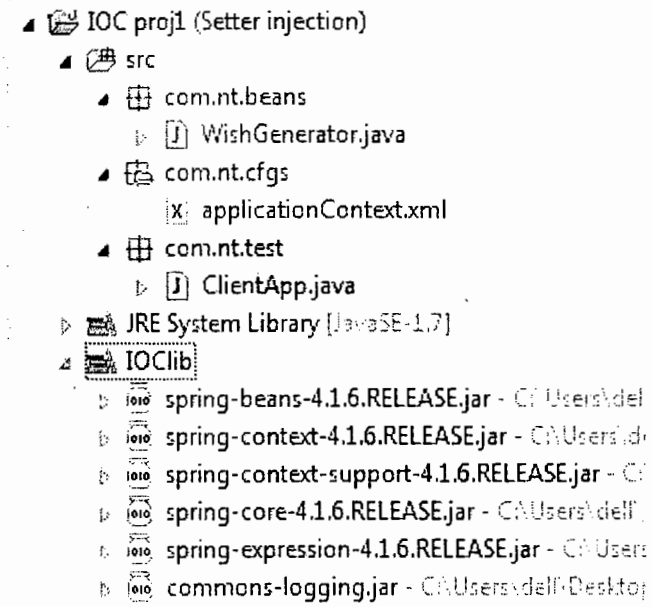
applicationContext.xml:

```
<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-3.0.xsd">
<bean id="dt" class="java.util.Date"/>
<bean id="wsg" class="com.nt.beans.WishGenerator>
<constructor-arg value="Raja"/>
<constructor-arg value="dt"/>
</beans>
```

Based on above configuration the spring container uses 0-param constructor to create Date class object and uses 2-param constructor to create WishGenerator class object. (because the *<constructor-arg>* tag is placed 2 times in the xml file). In this process injection also will be completed.

Procedure to develop spring core module app that performs setter injection:

Step 1: keep eclipse IDE ready
Step 2: Design the application



Step3: Develop the above resources
(Workspace : NtSp57)

Design Pattern:-
It is set of rules as best solution for reoccuring problems of Apllication development.
It is best practice to develop Software applications.

Factory Pattern:-
provides abstraction on Object creation process.It creats and returns object but does not expose how that object is created.
DriverManager.getConnection(--) is designed around factory pattern becoz it creates the objs for diff implementation classes and returns those objs as con objects based on the urls we pass.
BeanFactory is designed around Factory pattern becoz it abstracts the object creation process of spring bean and returns object based on the given bean id. While creating main bean class obj it creates injects Dependent bean class objs.

wishGenerator.java (pojo class):

```
package com.nt.beans;
import java.util.Date;
public class WishGenerator{
//bean properties
private String name;
private Date date;
//Getter methods for setter injection
public void setName( String name){
this.name=name;
}
public void setDate(Date date){
this.date=date;
}
//B.logic
public String generatorWishMsg (){
return "GoodMorning" +name+ "--->" +date;
}
}
```

applicationContext.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-3.0.xsd">
<bean id="dt" class="java.util.Date"/>
<bean id="wsg" class=com.nt.beans.WishGenerator">
<property name="name" value="Raja"/>
<property name="date" ref="dt"/>
</bean>
</beans>
```

Note:

XSD,DTD are the two technologies that can be used to define building blocks or rules for constructing xml documents.....

XSD-->Xml Schema Definition

ClientApp.java:

```
package com.nt.test;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.beans.WishGenerator;
public class ClientApp {
    public static void main(String[] args) {
            System.out.println("ClientApp:main(-)");
     //Locate Spring Bean configuration file
            FileSystemResource res=new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml");
     //Activate BeanFactory container
```

```
        XmlBeanFactory factory=new XmlBeanFactory(res);
    // Get Bean class object
        //WishGenerator bean=(WishGenerator)factory.getBean("wsg");
        WishGenerator bean=factory.getBean("wsg",WishGenerator.class);
    // Call B.method
        String result=bean.generateWishMsg();
    System.out.println(result);
    }//main
}//class
```

Step 4: Add the following jar files to Build path of project.

Right click on project-->Buildpath-->Config buildpath-->libraries--->Addexternal jars-->
Add external jars-->
Commons-logging.jar (collect from spring 2.5)
Spring-beans-<version>.jar
Spring-context-<vers>.jar
Spring-context-support-<version>.jar
Spring-core-<vers>.jar
Spring-expression-<version>.jar

Step5: Run the client Application

FileSystemResource res=new FileSystemResource("src/com/nt/cfgs/applicationContext.xml"**)**;

Gives resource object by using java.io.File (1);
Internally pointing to the path and filename of spring bean configuration file.

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

Based on the given "res" object it locates the spring bean configuration file (xml) from specified path of file system.
Checks whether that xml file is well formed and valid documents or not.

Note1: If Xml file satisifies springframework syntax rules then it is called well-formed document.
Note:2 If xml file satisifies the xsd and DTD rules then it is called valid document.

Creates an empty logical IOC container in jvm memory.
Uses the SAXpraser (xml parser) to read xml file (Bean configuration file) data and maintain it as inmemory metadata or inmemory configuration.
Returns XmlBeanFactory class object ref representing IOC container.
WishGenerator bean=(WishGenerator)factory.getBean("wsg");
a)Loads "com.nt.beans.WishGenerator" class from inmemory metadata based on the given "wsg" and creates object for it.
Class.forName("com.nt.beans.Wish").newInstance();

b)Observes "name" with *<property>* tag and calls setName("Raja") method on WishGenerator class object.

c)Observes "date" with <property> tag and load "java.util.Date" class to create the Object.
Class.forName("java.util.Date").newInstance();

Also setDate(Date Class object) on WishGenerator object to perform setter injection.

d)Keeps Date class object and WishGenerator class object in IOC container as elements of HashMap.

Key	***Value***
dt	*Date Class object (dt)*
wsg	*WishGenerator class object (wsg)*
(bean id)	*(Bean class object)*

e)Returns WishGenerator class object reference back to ClientApp

If we configure both setter injection and constructor injecton on bean property the values given by setter injection will remain final values of injection because *setter(-)* method executes after constructor executes and overrides the values injected by the constructor execution or injection.

If spring bean class is configured without any injections or only with setter injection (one or more properties) then container uses O-param constructor to create beans class object. If any property is configured for constructor injection then it uses parameterized constructor to create bean class object.

In spring 3.x bean properties and its setter methods can be taken as static. But in spring 2.x we cannot take setter methods as static.

WishGenerator bean=(WishGenerator)Factory.getBean("wsg");
(Typecasting mandatory)
WishGenerator bean=Factory.getBean("wsg", WishGenerator.class)
(we can avoid type casting)

Prototype:

public Object getBean (String id);
public <T> getBean(String id, <T>.class)

What is the difference between FileSystemResource and ClassPathResource?
FileSystemResource can locate given spring bean configuration file from the specified path of file system. Here we can pass either absolute path or relative path.
FileSystemResource res=new FileSystemResource ("src/con/nt/cfgs/applicationContext.xml")
ClasspathResource can locate given spring bean configuration file from directories or jarfiles that are added to classpath/buildpath.
ClassPathResource res=new ClassPathResource("applicationContext.xml");
(But add com.nt.cfgs pakage to build path)
Right click on project->buildpath-->library-->Add class folder---->.....
by default src folder of the project (eclipse)is automatticaly place in builed path class path we can also write as soon below
Resourse res=new ClassPathResource("com/nt/cfgs/applicationContext.xml");

Understanding "Collaboration" and tight coupling, loose coupling:

- In real project development we can't develop entire logic in one class. We need to take multiple classes and these classes will maintain dependency with each other.
- The Class that uses other class logics/method is called target class and the class that acts as helper class is called dependent class and we can say both classes are in collaboration.

- Filpkart and DTDC classes must be there in collaboration. i.e Filpkart class is target class/main class and DTDC class is dependent class.
- To make target class/main class using logics of dependent class we can use the following approaches.

Approach 1:
Create Dependent class object in targetclass.

Approach 2:
Creates dependent class object in Factory class and makess target class using that factory class.

Approach3:
Target class gets Dependent class object from jndi registry.

Approach 4:
Target class extends from dependent class.

Approach 5:
Make underlying container injecting dependent class object to target class.

Approach 1:
Here target class creates the object of dependent class uses that object.
This is nothing but composition (casses containg HAS -A relation)

Target class	***Dependent Class***
class Filpkart	*class DTDC*
{	*{*
DTDC dtdc=new DTDC ();	*deliver();*
public void purchage();	.....
dtdc.deliver();	*}*
}	*}*
}	

Limitations :
1) Instead of DTDC courier, if flipkart want to use BuleDart/FirstFlight courier, we need to modify target class (Flipkart).
2) If *deliver()* is changed to *supply()* method in DTDC class we need to modify the filpkart class.

The above limitation indicates target and dependent classes are tightly coupled.

Approach 2:
The class that can create and return one or more objects is called Factory class.

public class DTDCFactory{	*class DTDC{*
public static DTDC getInstance()	*public void deliver()*
{	*{*
return new DTDC();	.....
}	.....
}	*}*
class Filpkart	*}*
{	
public void purchase()	

```
{
DTDC dtdc=DTDCDFactory.getInstance();
dtdc.deliver();
}
}
```

Limitations:

1) If filpkart wants to uses BuleDart then it need BuleDartFactory class instead DTDCFactory class

2)Same as approach 1

Note: Heretightly couplingobserves.

Approach 3:

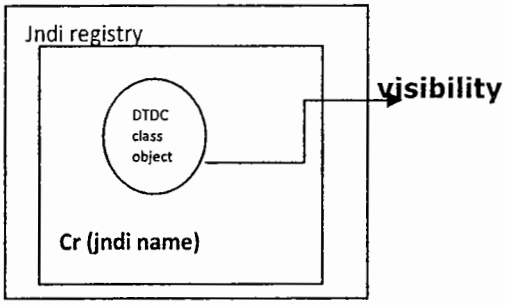
If you want to provide global visibility object or object reference then it should be placed in jndi registry.

eg: rmi registry, cos registry and etc.....

Note: Every webserver/application server gives one built-in jndi registry.

server

```
class Filpkart{
public void purchase(){
InitialContext ic=new InitialContrext();
  DTDC dtdl=(DTDC)ic.lookup("cr");
dtdl.deliver();
}
}
```



Advantage:

1)Because jndi registry DTDC object gets global visibility.

2)Filpkart class can use already available DTDC class object.

Limitation:

1)If Filpkart wants BuleDart object instead DTDC object we need to place that object in jndi registry.

2)Same as approach 1.

It also gives tight coupling.

if we create object in method . object is visible within the method.
if we create object in class as instanceof static variable then object is specific to that class of object to provide more globle visibility for the object.
use jndi registry.

Approach 4:

Make target class extending from dependent class.

```
class DTDC                          class Filpkart extends DTDC
{                                   {
deliver ()                          purchase()
  {                                 {
  ............                      deliver();
  }}                                      }}
```

Limitations:

a)Filpkart class cannot extends more than one dependent class.

b)In order to change dependency from DTDC to BuleDart, we need to makes Filpkart extending from BuleDart class.

c)Same as approach 1
Note:shows"*tight coupling*".

Approach 5:
Make underlying container injecting dependent object to target object.
Use dependency injection (setter injection/constructor injection)

```
class Filpkart
{
private DTDC dtdc;
//setter method
public void setDtdc(DTDC dtdc)
{
this.dtdc=dtdc;
}
 public void purchase ()
{
.....
.....
dtdc.deliver();
}
}
```

```
class DTDC
{

public void deliver()
{
.....
.....
}
}
```

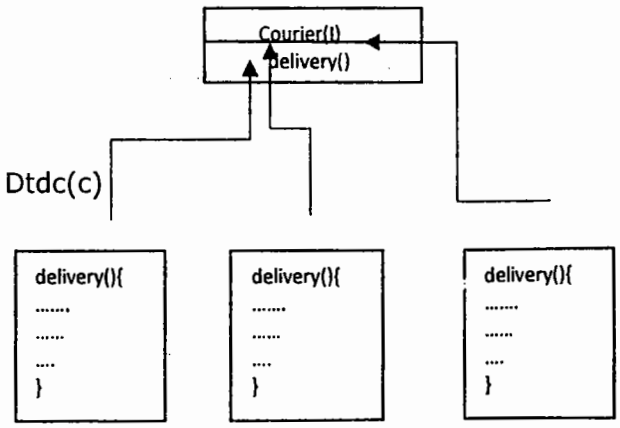
This approach 5 is much similar to approach 1 or approach 2.only difference is the underlying container is managing dependency in appraoch5 where as programmer is managing dependency in approach 1 and 2.

Limitations:
Same as approach 1

Note: shows tightly coupling.....

Approach 5 is good towards managing dependency. But in order to achive loose coupling we shouldn't directly work with dependent classes in target class. It is recommended to makes all dependent classes implementing common interface having common method declaration. So that we can start using this common interface in the target class

```
class Filpkart
   {
   private Courier courier;
   public void setCourier(Courier courier)
   {
   BlueDart(c)        FirstFlight(c)
   this.courier=courier;
   }
    public void purchase(){
   courier.delivery();
    .........}}
```



The above **"Filpkart(c)"** class is not working with implement class name. It is working with interface. So we can make underlying container injecting any implement class object to FlipKart (target class).

All dependent classes are implementing common interfaces "Courier(I)", so there is no possibility of changing method name "deleiver()"

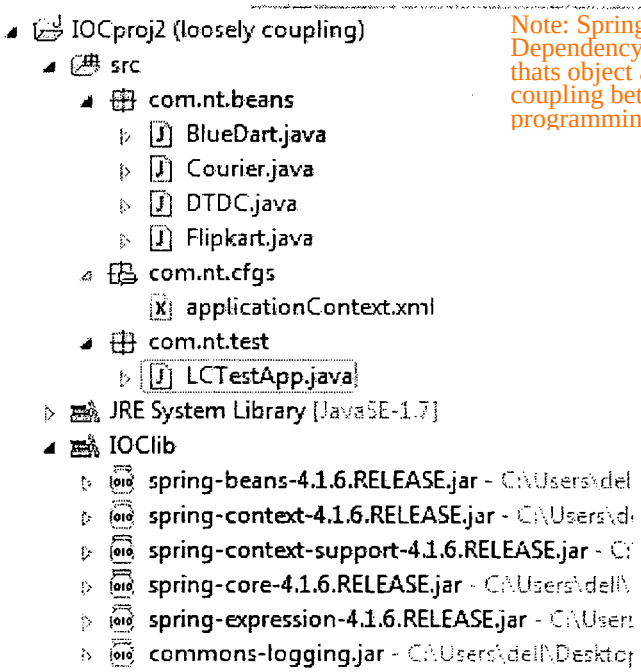
Note:

The above code shows lossely coupling between target class (Flipkart) and dependent classes.

To achive lossely coupling in spring use.

a)Dependency injection

b)Interface model programming (POJI model)



Note: Spring Dependency injection is good for bean collaboration or for Dependency management because its load both target and dependeds object, creates thats object and assigns one object with another object but to get loose coupling between classes it is recommnded to go for interface model programming.

Courier.java:

```
package com.nt.beans;
public interface Courier {
    public String deliver(int orderid);
}
```

BlueDart.java:

```
package com.nt.beans;
    public class BlueDart implements Courier {
        public BlueDart(){
        System.out.println("BlueDart:0-param constructor");
        }
        @Override
        public final String deliver(int orderid) {
                return "BlueDart is ready to deliver products of "+orderid;
        }
    }//class
```

DTDC.java:

```
    package com.nt.beans;
    public class DTDC implements Courier {
    public DTDC(){
```

```
            System.out.println("DTDC:0-param constructor");
        }
        public final String deliver(int orderid) {
            return "DTDC is ready to deliver products of "+orderid;
        }
    }//class
```

Flipkart.java:

```
package com.nt.beans;
import java.util.Random;
public class FlipKart {
    private Courier courier;
    public FlipKart(){
            System.out.println("FlipKart:0-param cosntructor");
    }
    //setter method for setter injection
    public void setCourier(Courier courier){
            System.out.println("FlipKart:setCourier(-)");
            this.courier=courier;
    }
    public String purchase(String items[]){
            //generate order id
            Random rad=new Random();
            int orderid=rad.nextInt(700000);
            // deliver order
            String status=courier.deliver(orderid);
            return status+ "Bill Amt for order id"+orderid+" is 7000";
    }//method
}//class
```

applicatioContext.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.xsd">
<bean  id="dtdc" class="com.nt.beans.DTDC" />
<bean  id="blueDart" class="com.nt.beans.BlueDart" />
<bean id="fpk" class="com.nt.beans.FlipKart" >
<property name="courier"><ref bean="dtdc"/></property>
</bean>
</beans>
```

LCTestApp.java:

```
package com.nt.test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.beans.FlipKart;
public class LCTestApp {
    public static void main(String[] args) {
      // Create BeanFactory object to create IOC contianer
      BeanFactory factory=new XmlBeanFactory
                                                (new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));

            // Get Bean object
            FlipKart  bean=factory.getBean("fpk",FlipKart.class);
            factory.getBean("fpk");
    String billmsg=bean.purchase(new String[]{"shirt","mobile","books"});
            System.out.println(billmsg);
    }//main
}//class
```

In spring bean config file:

```
<property name="msg" value="Hello"/>Can be written as
<property name="msg"><value>Hello</value></property>
<property name="date" ref="dt"/>Can be written as
<property name="date"><ref bean="dt"/> or <ref local="dt"/></property>
```

Attaching source code and api docs (Spring) with eclipse ide:

Right click on project --->buildpath--->configurebuildpath
Select any jar file that contains the classes that you are using

Source attachment

edit--->ecternal location----> path:--->external file-->select
<spring_home>/libs/spring_beans_<x.x>
Release_sources jar.file

Java doc location

Edit---->javadock in archive-->external file--->archive path (Select spring software zip file)
Path in archive : select..docs/javadoc_api folder in the above zip file --->validate-->ok

Creating user-defined library (SpringIOC) in eclipse IDE:

Right click project-->build path---->configure build path---->library---->Add library-->user library-------->next---->userlibrary--->new-->IOClibrary-->Add external jars--->select required jar files.

Design Patterns

Design patterns are set of rules which come as best solutions for reoccurring problem of application development.Design patterns are best practices for utilizing software languages, technologies andFrameworks effectively in application development.
eg: Singleton, factory and etc....
BeanFactory (IOC container) is designed based on the factory pattern....
It is always recommended to develop spring apps based on strategy pattern.

Stratgery pattern:

While developing multiple classes having dependencyit is recommended to follow this design pattern.This pattern is not spring pattern, it can be used anywhere since spring manages dependency between classes, it is recommended to use this pattern while developing spring application.Strategy pattern is all about implementing 3 principles.

a) Prefer composition(HAS-A) over inheritance.
b) Always code to interface and never code with implementations(classes)
c) Code should be open to extension and must be closed for modification

a)Prefer composition over inheritance:
If one class creates the objects of other class to use its logics then it is called composition

(Has –A Declarationship)

```
class A{
.............
............
}
class B
{
A a=new A();
}
```

If one class extends from another class then it is called inheritance.

```
class A
{
..........
......
}
class B extends A
{
...................
...................
}
```

<u>Inheritance is having following limitation in java</u>.
a)It doesn't support multiple inheritance
b)Code becomes easily breakable

```
class C extends A,B
{
.....
.....
}
class C {
A a=new A();
B b=new B();
}
```

Code is easily breakable:

eg:

```
class Test{
 public int x()
{
.....
.....
return 100;
}
}
class Demo extends Test {
public int x(){
.....
.....
return 1000;
}
}
```

If we change return type or param tyee of 'x()' in "Test" we can not compile "Demo" class because overriding rules (let us "Test","Demo" class given by 2 different developers)

Solution:

```
class A{
public int x(){
.....
return 100;
}
}
class B
{
A a=new A();
public int y(){
int res=a.x();
}
}
```

- If 'x()' method return type is changed in "A", then there is no need of doing majormodifications by in 'B'.
- If x() return type is changed to float then we need to write y() in 'B' as shownbelow

```
public int y(){
float res=a.x();
.........
.................
}
```

*The other classes using B class need not to have any modifications

b)Always code to interface and never code to implementation:

```
class A{
............
}
class B{
................
..........
}
class Test{
A a=new new A();
................
.............
}
```

- Asigning class "B" object instead of class "A" object in Test class is not possible with minimum modification.

eg:

```
interface x{
................
...................
}
class A{
................
................
}
class B implements x{
.........
........
}
class Test{
X x=new A();
Or
X x=new B();
.......................
...................
}
```

- When we code with interfaces we can achive loosly coupling between target class and dependent class.

```
interface X{
......
}
class A implements X{
........
}
class B implements X{
......
}
class Test{
X x;
public void setX(X x){
this.x=x;
}}
```

```
Test t=new Test();
t.setX (new A());
t.setX (new B());
```

Note:

Without modifying the code of "Test" class we can assign either the object for class A or class B. This gives loosly coupling.

Code must be open for extension and must be closed for modification.

If we follow second principle perfectly by taking the methods of implementation classes A,B (as method) our code becomes closed for modification.Similarly it allows to more implementation classes for interface x having new logics. This makes our code open for extension.

Observing strategy pattern in Application no:2 :

a)Flipkart class wants the logics of DTDC or BuleDart classes. But it is not inheriting from those classes, but getting object of those classes through composition. (principle 1)

BlueDart, DTDC classes are implementing "Courier(I)" or their objects are assigned to FliplKart class through Courier(I). (principle 2)

b) By making the deliver method of BuleDart,DTDC classes as final methods, we can keep that code class for modification and we can develop new implements class for "Courier(I)" if new Courier service (like firstflight) is required for Flipkart class (principle-3)

c) By making the 'delivery()' method of BlueDart,DTDC classes as final methods we can keep that code closed for modification and we can develop new implementation class for "Courier(I)". If new courier service like "FirstFlight" is required for "Flipkart" class (principle 3)

Understanding Realtime Dependecy Injection

DAO class-→ Java class that contains persistence logic
Service class-→ java class that contains B.logic
Controller class-→ Java class that receives request and deligates to Service class..

StudentVO.java

```
package com.nt.vo;
public class StudentVO {
	private String sno,sname,m1,m2,m3;

	public String getSno() {
		return sno;
	}

	public void setSno(String sno) {
		this.sno = sno;
	}

	public String getSname() {
		return sname;
	}

	public void setSname(String sname) {
		this.sname = sname;
	}

	public String getM1() {
		return m1;
	}

	public void setM1(String m1) {
		this.m1 = m1;
	}

	public String getM2() {
		return m2;
	}

	public void setM2(String m2) {
		this.m2 = m2;
	}

	public String getM3() {
		return m3;
	}

	public void setM3(String m3) {
		this.m3 = m3;
	}
}
```

StudentBO.java

```
package com.nt.bo;

public class StudentBO {
	private int sno;
	private String sname;
	private int total;
	private float avg;
	private String result;

	public int getSno() {
		return sno;
	}
	public void setSno(int sno) {
		this.sno = sno;
	}
	public String getSname() {
		return sname;
	}
	public void setSname(String sname) {
		this.sname = sname;
	}
	public int getTotal() {
		return total;
	}
	public void setTotal(int total) {
		this.total = total;
	}
	public float getAvg() {
		return avg;
	}
	public void setAvg(float avg) {
		this.avg = avg;
	}
	public String getResult() {
		return result;
	}
	public void setResult(String result) {
		this.result = result;
	}
}
```

StudentDTO.java

```
package com.nt.dto;
import java.io.Serializable;
public class StudentDTO implements Serializable {
	private int sno;
	private String sname;
	private int m1,m2,m3;
	public int getSno() {
		return sno;
	}
	public void setSno(int sno) {
		this.sno = sno;
	}
	public String getSname() {
		return sname;
	}
	public void setSname(String sname) {
		this.sname = sname;
	}
	public int getM1() {
		return m1;
	}
	public void setM1(int m1) {
		this.m1 = m1;
	}
	public int getM2() {
		return m2;
	}
	public void setM2(int m2) {
		this.m2 = m2;
	}
	public int getM3() {
		return m3;
	}
	public void setM3(int m3) {
		this.m3 = m3;
	}
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName"  value="oracle.jdbc.driver.OracleDriver"/>
<property name="url"  value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
```

```
<property name="password" value="tiger"/>
<property name="connectionProperties">
<props>
<prop key="user">scott</prop>
<prop key="password">tiger1</prop>
</props>
</property>
</bean>
<bean id="stDao" class="com.nt.dao.StudentDAOImpl">
<constructor-arg  ref="drds"/>
</bean>
<bean id="stService" class="com.nt.service.StudentServiceImpl">
<constructor-arg ref="stDao"/>
</bean>
<bean id="stController" class="com.nt.controller.StudentController">
<constructor-arg ref="stService"/>
</bean>
</beans>
```

Here we are injecting values to the Property ConnectionProperties of type java.util.properties. If username, pwd properties are set explicitly they will override the content of connection properties.

StudentController.java

```
package com.nt.controller;

import com.nt.dto.StudentDTO;
import com.nt.service.StudentService;
import com.nt.vo.StudentVO;

public class StudentController {
	private StudentService service;

	public StudentController(StudentService service) {
		this.service = service;
	}

	public String process(String sno,String sname,String m1,String m2,String m3){
	 //prepare VO object
		StudentVO vo=new StudentVO();
		vo.setSno(sno);vo.setSname(sname);
		vo.setM1(m1);vo.setM2(m2);vo.setM3(m3);
		//Convert to VO object to DTO object
		StudentDTO dto=new StudentDTO();
		dto.setSno(Integer.parseInt(vo.getSno()));
		dto.setSname(vo.getSname());
		dto.setM1(Integer.parseInt(vo.getM1()));
		dto.setM2(Integer.parseInt(vo.getM2()));
		dto.setM3(Integer.parseInt(vo.getM3()));
		//use Service class
		String result=service.generateResult(dto);
		return result;
	}//method
}
```

StudentService.java

```
package com.nt.service;

import com.nt.dto.StudentDTO;
public interface StudentService {
	public String generateResult(StudentDTO dto);

}
```

StudentServiceImpl.java

```
package com.nt.service;

import com.nt.bo.StudentBO;
import com.nt.dao.StudentDAO;
import com.nt.dto.StudentDTO;

public class StudentServiceImpl implements StudentService {
	private  StudentDAO  dao;

	public StudentServiceImpl(StudentDAO dao) {
		this.dao = dao;
	}
	@Override
	public String generateResult(StudentDTO dto) {
		//write business logic
		int total=dto.getM1()+dto.getM2()+dto.getM3();
		float avg=(float)total/3.0f;
		String result=null;
		if(avg<35)
			result="Fail";
		else
			result="Pass";
		//create BO class object
		StudentBO bo=new StudentBO();
		bo.setSno(dto.getSno());
		bo.setSname(dto.getSname());
		bo.setTotal(total);
		bo.setAvg(avg);
		bo.setResult(result);
		//use DAO
		int cnt=dao.insert(bo);

		if (cnt==0)
			return  "Result:"+bo.getResult()+"        "+bo.getSno()+"  Registration
Failed";
		else
			return  "Result:"+bo.getResult()+"        "+bo.getSno()+"  Registration
Succedded";
	}//method
}//class
```

StudentDAO.java

```
package com.nt.dao;

import com.nt.bo.StudentBO;

public interface StudentDAO {
	public int insert(StudentBO bo);

}
```

StudentDAOImpl.java

```
package com.nt.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.sql.DataSource;

import com.nt.bo.StudentBO;

public class StudentDAOImpl implements StudentDAO{
	private static final String STUDENT_INSERT_QUERY="INSERT INTO SP_STUDENT
VALUES(?,?,?,?,?)";

	private DataSource  ds;
	public StudentDAOImpl(DataSource ds) {
		this.ds = ds;
	}

	@Override
	public int insert(StudentBO bo) {
		//write jdbc code to insert record
		try{
			//get jdbc con
			Connection con=ds.getConnection();
			//create PreparedStatement object
			PreparedStatement
ps=con.prepareStatement(STUDENT_INSERT_QUERY);
			// set Query param values
			ps.setInt(1,bo.getSno());
			ps.setString(2,bo.getSname());
			ps.setInt(3,bo.getTotal());
			ps.setFloat(4,bo.getAvg());
			ps.setString(5,bo.getResult());
			// execute the Query
			int result=ps.executeUpdate();
			return result;
		}
		catch(SQLException se){
```

```
                return 0;
            }
            catch(Exception e){
                return 0;
            }
        }//method
}//class
```

ClientApp.java

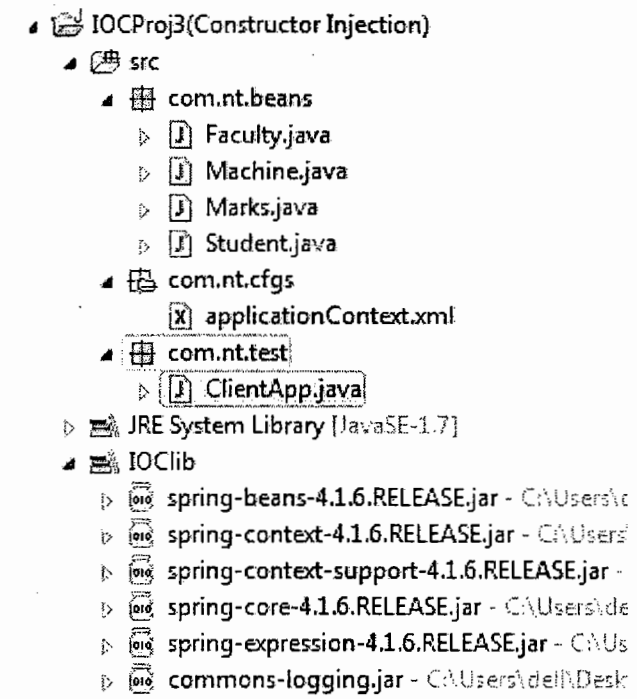
```
package test;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

import com.nt.controller.StudentController;

public class ClientApp {

        public static void main(String[] args) {
            //Create IOC Container
BeanFactoryfactory=newXmlBeanFactory(new
FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
            // Get Bean
            StudentController
controller=factory.getBean("stController",StudentController.class);
            //call method
            System.out.println(controller.process("1013","raja","56", "67","78"));
        }
}
```

Resolving/identifying params in constructor injection:

We generally configure the dependent values of constructor injection in the order the params of constructor are available. But we can mismatch this order during configuration if we can identify the params by using `index or type or name`.

eg:

Resource based parameter type:



Student.java:

```
package com.nt.beans;
public class Student {
    private int sno;
    private String sname;
    private float avg;
    public Student(int sno, String sname, float avg) {
      System.out.println("Student:3-param constuctor");
            this.sno = sno;
            this.sname = sname;
            this.avg = avg;
    }
    @Override
    public String toString() {
    return "Student [sno=" + sno + ", sname=" + sname + ", avg=" + avg + "]";
    }
}
```

applicationContext.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.xsd">

<bean id="St" class="com.nt.beans.Student">
<constructor-arg value="1001" type="int"/>
<constructor-arg value="89.45F" type="float"/>
<constructor-arg value="raja" type="java.lang.String"/>
</bean>
</beans>
```

Limitation:

Ifmutliple params are having same type then this approach cant be use.

EX2: Resolving/identifying based on index:

Marks.java

```
package com.nt.beans;
public class Marks{
      private int m1,m2,m3;
      public Marks(int m1,int m2,int m3){
        this.m1=m1;
        this.m2=m2;
        this.m3=m3;
       }
      //Place toString()
     @Override
     public String toString() {
            return "Marks [m1=" + m1 + ", m2=" + m2 + ", m3=" + m3 + "]";
     }
}
```

applicationContext.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.xsd">
<bean id="mk" class="com.nt.beans.Marks">
<constructor-arg value="30" index="2"/>
<constructor-arg value="20" index="1"/>
<constructor-arg value="10" index="0"/>
</bean>
</beans>
```

Ex3: Resolving configure params through name:

Faculty .java:

```
package com.nt.beans;
import java.beans.ConstructorProperties;
public class Faculty {
    private int id;
    private String name,subject;
    @ConstructorProperties({"no","fname","sub"})//optional in spring 4.x
    public Faculty(int no, String fname, String sub) {
            this.id = no;
            this.name = fname;
            this.subject = sub;
    }
    public String toString() {
    return "Faculty [id=" + id + ", name=" + name + ", subject=" + subject+ "]";
    }}//class
```

while working with setter injection name we specify in the name attribute of property tag is not property name . it is Xxx word of setXxx method .

while similarly in constructor injection the name we specify in the name attribute of constructor -arg tag is not property name that is param name.

applicationContext.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.xsd">
    <bean id="st" class="com.nt.beans.Student">
    <constructor-arg value="1001" name="no"/>
    <constructor-arg value="java" name="sub"/>
    <constructor-arg value="raja" name="fname"/>
    </bean>
    </beans>
```

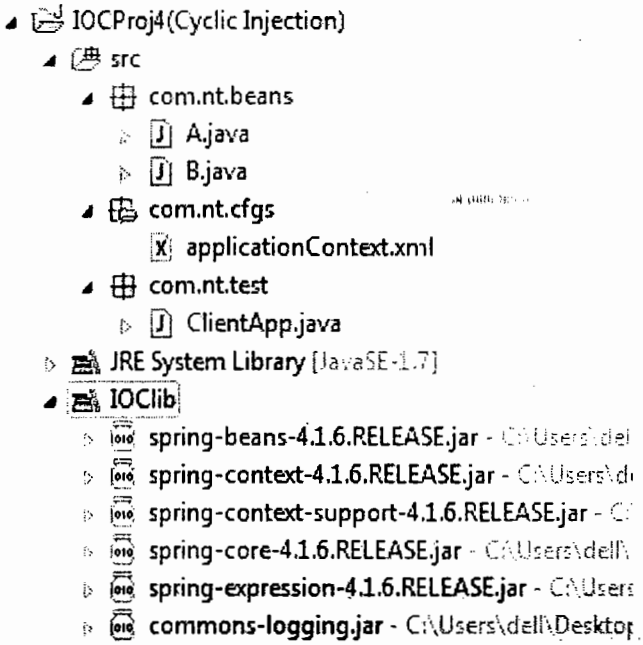
Conclusion : In real time resolving constructor parameter using different concepts technique will not be used. only technique that will be used is configuring value to constructor params in the order they are available in the constructor.

- In **Spring 3.x** the IOC container can't get the parameter names of constructor directly until the parameter names are configured by using "**@ConstructorProperties(-)**".
- In spring 4.x the IOC container can recoganise constructor parameters directly.

Cyclic/Circular dependency injection:

If two beans are dependent to each other then we can say they are in cyclic dependency. Spring container can't perform cyclic dependency injection through constructor injection but can perform through setter injection.

Problem:




```
class A{
B b;
 public A(B b){
this.b=b;
}}
class B{
A a;
public B(A a){
this.a=a;}
}
```

applicationContext.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.springframewrk.org/schema/beans/spring-beans.xsd">
    <bean id="a" class="A">
    <constructor-arg ref="b"/>
    </bean>
    <bean id="b" class="B">
    <constructor-arg ref="a"/>
    </bean>
    </beans>
```

***Solution**: (Using Setter injection)*

```
class A{
B b;
this.b=b;
}}
class B {
A a;
public void setA(A a){
this.a=a;
}}
```

applicationContect.xml (Supports)

```
<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.xsd">
<bean id="a" class="A">
<property name="b" ref="b"/>
</bean>
<bean id="b" class="B">
<property name="a" ref="a"/>
</bean>
</beans>
```

When should we go for setter injection and when should we go for constructor injection?

Ans: If bean class contains only one property or if all the property of bean class should participate in dependency injection then go for constructor injection, because construct injection is very faster than setter injection.

If Bean class contains more than one property and there is no need of making all properties participating in dependency injection, then we need to go for setter injection.

eg: If bean class contains four properties and there is no need of making all properties participating in dependency injection, then we need to write max of four setter methods to perform setter injection. But we need to write four factorial (24) constructors to support constructor injection in all angles.

Difference between setter injection and constructor injecton:

Setter injection	Constructor injection
Performs injection after creating Bean class object so it is bit delayed injection.	Performs injection while creating Bean class object so no delay in injection.
Supports cyclic dependency injection	Doesn't supports cyclic injection
<property> tag is required.	<constructor-arg> tag is required.
To perform setter injection on "n-properties" In all angles "n-setter" methods are required.	To perform constructor injection on n properties in all angles n! constructors are required.
If all properties are configured for setter Injection then container creates bean class object using O-param constructor.	If any property is configured for constructor injection container bean class object using parameterized constructor.

Inner Bean:

This is no way related with inner classes of java. It is all about configuring one bean as inner bean of another bean by placing *<bean>* tag inside the *<property> or <constructor-arg>* tag.

eg:

```
<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.xsd">
<bean id="fpk" class="com.nt.beans.Flipkart">
<property name="courier">
<bean id="dtdc" class="com.nt.beans.DTDC"/>
</property>
</bean>
</beans>
```

Use inner bean configuration if we want to make bean as depedent bean to another bean only for one time.

- while developing webapp the validations are specific to each controller that process as each form generated so always cfg validate bean as dependent and innerbean to controllerbean. we can also cfg innerbean under <constructor-arg>.

Limitations with inner bean:

a) Inner bean cannot be used as dependent bean of multiple other beans.
b) Inner bean object cannot be accessed from client applications.

Note: Bean id is optional for inner beans.

Can we configure spring bean class without bean id?

Ans: Yes possible, because the container generates default bean 'id'inthe following notation.

<pkg>.<BeanClass>#n

#n--->0,1,2,3.....

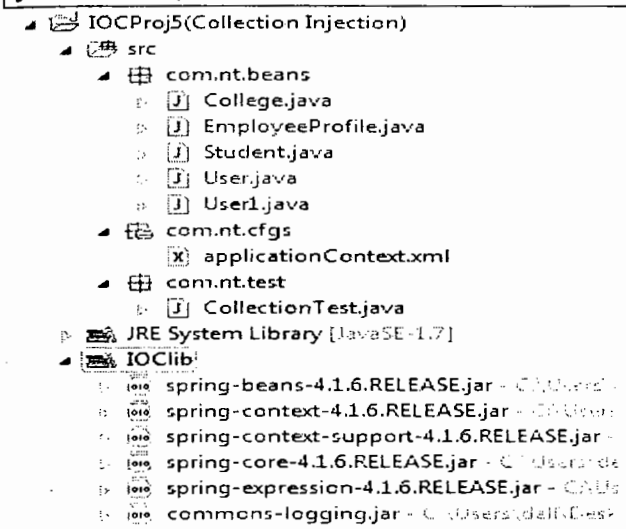
Assignment :can be take private class as springbean.
can be place only private constructor in springbean class

```
eg: <bean class="com.nt.beans.FlipKart">(bean id is "com.nt.beans.Flipkart#0")
.....................................
.......................
</bean>
<bean class="com.nt.Flipkart">(bean id is "com.nt.beans.Flipkart#1)
...........................
...................
</bean>
```

Spring supports dependency injection on the following properties

a) Simple properties (primitive data types, String)
b) Reference/Object type properties
c) Collection properties (Array, List, Set, Map, Properties, and etc.....)

Property type	Tag for dependency injection configuration
Simple property	value attribute or <value> tag
Reference property	ref attribute of <ref> tag
Array	<list> tag
java.util.List	<list>
java.util.Set	<set>
java.util.Map	<map>
java.util.Properties	<props>



Student.java:

```
package com.nt.beans;
import java.util.Arrays;
import java.util.List;
public class Student {
    private int marks[];
    private List<String> names;
      public void setNames(List<String> names){
          this.names=names;    }
public void setMarks(int[] marks) {
          this.marks = marks;
    }
    @Override
    public String toString() {
```

allows size is fixed size & needs memory continuously.
In Java Array is an obj having for dynamically generated in memory java class.
Collect^n are dynamically growable, no fixed size. Collect^n element doesn't allocate memory continuously.

```
        return "Student [marks=" + Arrays.toString(marks) + ", names=" + names
                                + "]";
    }
}
```

applicationContext.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.xsd">
    <bean id="st" class="com.nt.Student">
    <property name="colors">
    <list>
    <value>red</value>
    <value>green</values>
    </list>
    </property>
    </bean>
    </beans>
```

Internal Code:

```
Student st=(Student)Class.forName("Student").newInstance();
String colors[]=new String[]{"red","green"};
st.setColors(colors);
```

Injecting List:

```
public class Student{
private List<String> names;
public void setNames (List<String> names){
this.names=name;
}
//toString
...............
.............
}
```

applicationContext.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.xsd">
    <bean id="St" class="com.nt.beans.Student">
    <property name="names">
    <list>
    <value>Raja</value>
    <value>Ravi</value>
    <value>Raju</value>
    </list>
    </property>
    </bean>
    </beans>
```

Note: *java.util.List* allows duplicates. So we can inject duplicates values.

Internal code:

```
Student st=(Student)Class.forName("com.nt.beans.Student").newInstance();
List <String>names=new ArrayList <String>();
names.add("Raja");
names.add("Ravi");
names.add(Raju);
st.setName(names);
```

Injecting Set:

Java.util.Set type collection doesn't allows duplicate vales.

eg:

```
package com.nt.beans;
import java.util.Set;
public class User{
   private Set<String> phones;
  public void setPhones(Set<String> phones){
     this.phones=phones;
   }
  // toString()
@Override
  public String toString() {
    return "User [phones=" + phones + "]";
   }
 }
```

applicationContext.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.xsd">
<bean id="user" class="com.nt.beans.User">
<property name="phones">
<set>
<value>+91-9859922331</value>
<value>+91-892149322</value>
</set>
</property>
</bean>
</beans>
```

Injecting Map:

Each element of map allows one key, one value use <map> to configure injection.

```
eg:
package com.nt.beans;
import java.util.Map;
public  class  College{
       private Map<String,String> facultySubjects;
    private Map<?,?> facultyPhones;
      public  void setFacultySubjects(Map<String,String> facultySubjects){
          this.facultySubjects=facultySubjects;
```

```
        }
public void setFacultyPhones(Map<?, ?> facultyPhones) {
                this.facultyPhones = facultyPhones;
        }
        @Override
        public String toString() {
                return "College [facultySubjects=" + facultySubjects
                        + ", facurtyPhones=" + facultyPhones + "]";
        }
}
```

applicationContext.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.xsd">
<bean id="clg" class="com.nt.beans.College">
<property name="facuktySubjects">
<map>
<entry key="Raja" value="java"/>   [element-0]
<entry>   Key   value
<key><value>Ravi</value></key>   [element-1]
<value>.NET</value>
</entry>
<entry key="Ramesh">   [element-2]
<value>SAP</value>
</entry>
</map>
</property>
</bean>
</beans>
```

- Map collections elements allow any objects as key and any object as values, but keys cant be duplicated and values can be duplicated.

Injecting java.util.Properties:

- Java.util.Properties allows only String as the key-value pairs i.e key must be String and value must be String.

eg:

EmployeeProfile.java

```
package com.nt.beans;
import java.util.Properties;
public class EmployeeProfile{
    private Properties  empDesgs;
    public void setEmpDesgs(Properties empDesgs){
      this.empDesgs=empDesgs;
    }
    // toString()
  @Override
  public String toString() {
        return "EmployeeProfile [empDesgs=" + empDesgs + "]";
```

```
    }
  }
```

applicationContext.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.xsd">
  <bean id="emp" class="com.nt.beans.employeeProfile">
  <property name="empDesgs">
  <props>
  <prop key="Anil">Clerk</prop>   [prop-0]
  <prop key="Jani">SE</prop>
  key value
  </props>
  </property>
  </bean>
  </beans>
```

Injecting Null (<null>)

- Given to injection null value to reference type bean property.
- Useful in constructor injection to inject null value to certain reference type parameter when we cannot supply value.
- In constructor injection we need to supply values to all the params. If we cannot supply values to certain parameters then we can <null /> tag to inject null value.

eg: • we can't perform on simple type properties.

```
public class User{
private int id;
private String name;
private Date doj;
public user(int id, String name, Date dob){
this.id=id;
this.name=name;
this.dob=dob;
}
//toString()
.....
.....
}
```

can be place only spring pvt constructor?
Yes possible . Ioc container internally uses reflection api to access private constructor and instantiate private bean dynamically in run time.

```
Class c=Class.forName("com.nt.beans.Apartment");
Constructor cons=c.getDeclaredConstructors()[0];
cons.setAccessible(true);
cons.newInstance();
```

applicationContect.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.xsd">
<bean id="user" class="pkg.user">
<constructor-arg value="100"/>
<constructor-arg value="Raja"/>
<constructor-arg><null/></constructor-arg>
<!--null injection-->
</bean>
</beans>
```

Bean Inheritance

We use inheritance for reusability and extensibility. Bean inheritance is no way related with class level physical inheritance. It is all about about xml file levelbean configurations inheritance.

When we want configure same class for multiple times as multiple spring beans we can use resuse of bean properties configuration across the multiple beans configuration by using bean inheritance.

Realtime example:

```
<bean id="drds" class="pkg.DriverManagerDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="system"/>
<property name="password" value="manager"/>
</bean>
<bean id="scds" class="pkg.SingleConnectionDataSource"/>
</beans>
```

parent = "drds"

(or)

<bean id = "bds" class = "pkg.BasicDataSource" parent = "drds" />

eg:-

- IOCProj6(Bean Inheritance)
  - src
    - com.nt.beans
      - Bike.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.test
      - ClientTest.java
  - JRE System Library [JavaSE-1.
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\d
    - spring-context-4.1.6.RELEASE.jar - C:\Users\
    - spring-context-support-4.1.6.RELEASE.jar -
    - spring-core-4.1.6.RELEASE.jar - C:\Users\del
    - spring-expression-4.1.6.RELEASE.jar - C:\Us
    - commons-logging.jar - C:\Users\dell\Deskt

Bike.java:

```
package com.nt.beans;
public class Bike{
  private  String id;
   private  String make;
  private  String engineCC;
 //write seTxx(-) methods
public void setId(String id) {
    this.id = id;
}
public void setMake(String make) {
    this.make = make;
}
public void setEngineCC(String engineCC) {
this.engineCC = engineCC;
}
  //write toString()
 @Override
 public String toString() {
    return "Bike [id=" + id + ", make=" + make + ", engineCC=" + engineCC + "]";
  }
}
```

applicationContext.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.xsd">
<bean id="pulsor1" class="com.nt.beans.Bike">
<property name="id" value="TS081111"/>
<property name="maxe" value="Bajaj"/>
<property name="engineCC" value="150"/>
</bean>
<bean id="pulsor2" class="com.nt.beans.Bike">
<property name="id" value="TS082222"/>
<property name="maxe" value="Bajaj"/>
<property name="engineCC" value="150"/></bean></beans>
```

Problem:

Make, engineCC property values configuration is not reusable even though they are same in multiple bean configurations.

Solution: (Bean inheritance)

applicationContext.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.xsd">
<bean id="pulsor1" class="com.nt.beans.Bike">
<property name="id" value="TS081111"/>
<property name="maxe" value="Bajaj"/>
<property name="engineCC" value="150"/>
</bean>
<bean id="pulsor2" class="com.nt.beans.Bike parent="pullsor1">
<property name="id" value="TS082222"/>
</bean>
</beans>
```

Solution2: *(improved Solution)*

```
<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.xsd">
<bean id="basepulsor" class="com.nt.beans.Bike" abstract="true">
<property name="maxe" value="Bajaj"/>
<property  name="engineCC" value="150"/>
</bean>
<bean id="pulsor1" class="com.nt.beans.Bike" parent="basepulsor">
<property name="id" value="TS021111"/>
</bean>
<bean id="pulsor2" class="com.nt.beans.Bike" parent="basepulsor">
<property name="id" value="Ts022222"/>
</bean>
</beans>
```

Important Points:

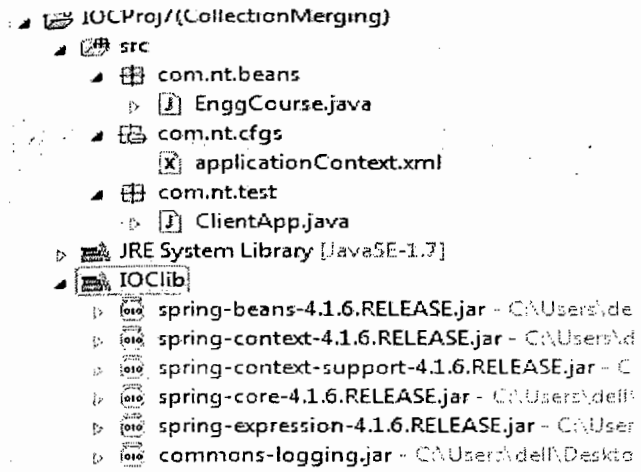
- It is not class level physical inheritance.
- Parent bean configuration and child bean configuration can refer same class or different classes.

Note:

When *abstract ="true"* it doesn't make bean class as abstract class, but it makes bean configuration as abstract bean configuration i.e we can't call *factory.getBean(-)* having that abstract bean configuration bean id (throws Exception)

Collection Merging

It allows to merge the value set to "Base bean" configuration collection property with the values set to "Child bean" configuration collection property by using ***merge="true"***attribute.



EnggCourse.java:

```
package com.nt.beans;
import java.util.List;
public class EnggCourse{
      private List <String> subjects;
     public void setSubjects(List <String>subjects){
       this.subjects=subjects;
       }
     // toString()
     @Override
     public String toString() {
            return "EnggCourse [subjects=" + subjects + "]";
     }
     }
```

applicationContext.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.xsd">
   <bean id=base1styear" class="com.nt.beans.EnggCourse" abstract="true">
   <property name="subjects">
   <list>
   <value>C</value>
   <value>Math</value>
   </list>
   </property>
   </bean>
   <bean id="ECE1styear" class="com.nt.beans.EnggCourse" parent="base1styear">
   <property name="subjects">
   <list>
   <value>digital elect</value>
   <value>C</value>
```

```
<value>Math</value>
</list>
</property>
</bean>
</beans>
```

Problem: Writing common Subjects (C,Maths) again in child bean config indicates duplication.

Approach2 :

applicationContext.xml: (Write this code along with base bean cfg of above code)

```
<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.xsd">
<bean id="ECE1st year" class="com.nt.beans.EnggCourse" parent="base1styear">
<property name="Subjetcs">
<list>
<value>digital Eletronics</value>
</list>
<property>
</bean>
</beans>
```

Problem:

Child bean configuration overrides the value of parent bean configuration.

Approach3: (add this code along with above code)

applicationContext.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.xsd">
<bean id="ECE1styear" class="EnggCourse" parent="Base1styear">
<property name="Subjecst">
<list merge="true">
<value>Dig EC</value>
</list>
</property>
</beans>
```

Important points:

i) Merging is possible only on collection type property.

ii) Base bean and Child bean configuration types must match to perform merging.

property type and names

Client Application:

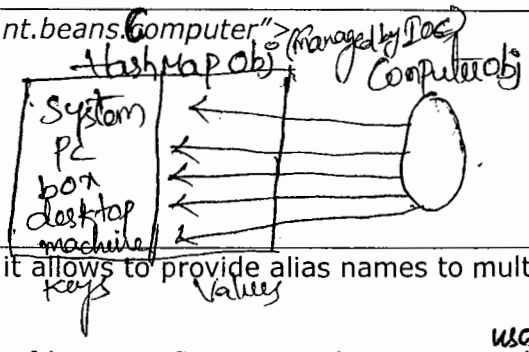
```
EnggCourse EcEengg=factory.getBean("ECE1styear", EnggCourse.class);
System.out.println (ECEengg);
```

Bean Alias

Alias name is nick name or petname. It is all about providing more alternative names/id's to use bean.Prior to spring 2.0 "name" attribute is given to provide "aliasname" from spring2.0 <alias> tag is introduced for the same.We can use both name attribute, <alias> tags to provide alias names from spring2.0.

eg:

```
<bean id="system" name="pc, desktop" alias="com.nt.beans.Computer">
.........................
.................
</bean>
<alias name="system" alias="machine"/>
<alias name="system" alias="box"/>
```



Note1: *<alias>* tag is recommended to use because it allows to provide alias names to multiple beans in single place.

Note2: aliasing is useful in the maintianance mode of bean configuration where we can use short alias name for lengthy beans ids.

//real time use of Bean Alias

Generally bean class names will be taken as bean id's,some times they will be quite lengthy to provide shotter version of lengthy bean id's we can take the support of bean alias name.

In ClientApp

```
factory.getBean("box");
factory..getBean("PC");
factory.getBean("System");
```

Performing dependency lookup by using IOC container:

If target class/main class writes the code to search and gather dependent values/objects is called dependency lookup or dependency pull. By creating IOC container in target class or main class we can perform dependency lookup/pull operation to get dependent objects.

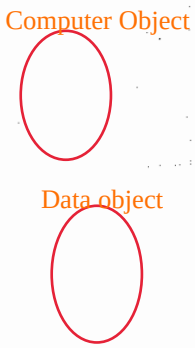
If target class is wants to use dependent object in all the methods of target class then go for dependency Inection.If target class wants to use dependent class object only in specific method then go for dependency lookup or pull.

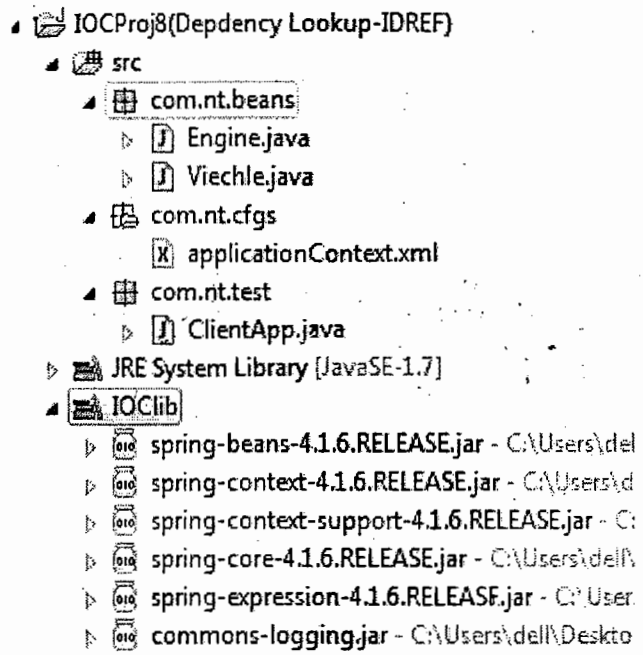
Spring supports both dependency injection and dependency lookup and we can use both concepts in single application.

If target bean class of spring application wants to use external services components gathered from JNDI register (like gathering score component reference, weather report component reference and etc then go for dependency lookup).

IOC Container HashMap

pc	system(ref variable)
system	system
desktop	system
machine	system
box dt	system dt(ref variable)
keys	values





Engine.java:

```
package com.nt.beans;
public class Engine{
    public Engine(){
            System.out.println("Engine:0-param constructor");
    }
  public void start(){
    System.out.println("Engg started");
  }}
```

Viechle.java:

```
package com.nt.beans;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.cglib.proxy.Factory;
import org.springframework.core.io.FileSystemResource;
public class Viechle{
 private  String enggId;
 public Viechle(){
     System.out.println("Viechle:0-param constructor");
 }
public void setEnggId(String enggId){
  this.enggId=enggId;
 }
public void move(){
   BeanFactoryfactory=newXmlBeanFactory(new
     FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
   Engine engine=factory.getBean(enggId,Engine.class);
   engine.start();
   System.out.println("Viechle  moved for journey");
 }
}
```

applicationContext.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.xsd">
<bean id="engg1" class="com.nt.beans.Engine"/>
<bean id="viechle" class="com.nt.beans.Viechle" >
<property name="enggId" value="engg1">
</bean>
</beans>
```

injects beanId as a normal String value

Client Application:

```
package test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.beans.Viechle;
public class ClientApp {
    public static void main(String[] args) {
            BeanFactory factory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
            Viechle viechle=factory.getBean("viechle",Viechle.class);
             viechle.move();
    }}
```

In the above code:

<property name="enggId" value="engg1"/> tag injects dependent class bean id (engine class) to target class object (vehicle class) as an ordinary 'String' value without checking whether bean class is configured or notwith that bean id. To overcome this problem and to inject bean-id only when bean class is configured with that bean-id we need to use '*<idref>*' tag.

```
eg:<bean id ="engg" class="com.nt.beans.Engine"/>
<bean id ="Vehicle" class="com.nt.beans.vehicle">
    <property name="enggId">
    <idref bean="engg1"/>===>
    <property>
    </bean>
```

Here the bean id "engg1" will be injected to the property enggId only as beanId not as String value.

eg: String name="test" ---->"test" class is maintained are String content.
Class name=Test.class-->"Test" class is maintained as class in the object of java.lang.Class

- Always prefer working with <idref> tag to inject beanIds to the properties of bean class beez it checks wether those beanIds are available or not in spring bean cfg file before inject^n.
- In the above applicat^n dependency lookup is happened but it is not the best way of doing that work beez we are taking 2nd IOC Container in target class method unnecessarily to overcome this problem use aware inject^n or lookup method inject^n.

24/12/16

FactoryMethod

The method that's capable of creating either same class object or different class object is called FactoryMethod.

Two types of factory methods

1) Instance factory method

Eg1:*String S1=new String ("Hello");*
String S2=S1.concat("123");//hello 123 (return same class object)

Eg2:*StringBuffer sb=new StringBuffer("Hello how are you?");*
String S1=sb.substring(0,4);//given hello (return different class object)

2) Static Factory method:

Eg1:*Class e= Class.forName("java.util.Date");* (returns same class object)
Eg2:*Console cons=System.console();*(returns different class object)

We can make IOC container to create spring bean class Object in 4 ways

1) Using 0-param constructor. (when all bean properties are cfg for setter injectn / no properties are cfg for const injectn)
2) Using parameterized constructor. (when any properties cfg for const injectn)
3) Using instance factory method. (when factory-bean, factor-method attributes of <bean> tag are used)
4) Using static factory method. when factory-method attribute of <bean> tag is used)

Note points:

>Servlet container ,jsp container can create bean class object by using 0-param constructor.
>Spring IOC container can create bean class object by using above 4 ways.
>In spring we can configure abstract class as spring bean only when we enable
a)Factory method bean instantiation
b)Lookup method injection

eg:

- IOCProj9(Factory methods)
  - src
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\del
    - spring-context-4.1.6.RELEASE.jar - C:\Users\d
    - spring-context-support-4.1.6.RELEASE.jar - C:
    - spring-core-4.1.6.RELEASE.jar - C:\Users\dell\
    - spring-expression-4.1.6.RELEASE.jar - C:\Users
    - commons-logging.jar - C:\Users\dell\Desktop

we can use constructor-arg tag not only for constructor injection configuration it also usefull to configure factory method argument value while performing factory methods bean instantiation.
while performing factory method bean instantiation configuration there is no need of writing class attribute.
when we unable static factory method bean instantiation we can take abstract class as bean class.

while performing static factory method bean instantiation the factory method return object will act as spring bean having given bean id.
while working with instance factory method bean instance the factory method is called given factory-bean attribute based bean class object and generated object become spring bean with the given bean id

- when we cfg factory() for bean instantiation we can use <constructor-arg> to supplay factory method argument values that means we can use <const-arg> for 1) constructor inject cfg
2) To supplay factory() arg values.

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instances
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<!-- Using static factory methods -->
<bean id="c1" class="java.lang.Class" factory-method="forName">
<constructor-arg value="java.util.Date"/>
</bean>
<bean id="c2" class="java.util.Calendar" factory-method="getInstance"/>
<!-- Using instance factory methods-->
<bean id="s1" class="java.lang.String">
<constructor-arg value="hello"/>
</bean>
<bean id="s2" factory-bean="s1" factory-method="concat">
<constructor-arg value="how are u?"/>
</bean>
<bean id="sb" class="java.lang.StringBuffer" >
<constructor-arg value="hello how are u?"/>
</bean>
<bean id="s3" factory-bean="sb" factory-method="subString">
<constructor-arg value="0"/>
<constructor-arg value="5"/>
</bean>
</beans>
```

cfg for constructor injects.

cfg to supplay factory() argument values.

ClientApp.java

```
package test;
import java.util.Calendar;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
public class ClientApp {
    public static void main(String[] args) {
        // Create IOC Container
        BeanFactory factory=new XmlBeanFactory(new
FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
            // get Bean
        Class c1=factory.getBean("c1",Class.class);
        System.out.println("c1 data"+c1+" c1 class name"+c1.getClass());
        //get Bean
        Calendar c2=factory.getBean("c2",Calendar.class);
        System.out.println("c2 data"+c2+" c2 class name"+c2.getClass());
        // get Bean
        String s2=factory.getBean("s2",String.class);
        System.out.println("s2 data"+s2+" s2 class name"+s2.getClass());
        //get Bean
        String s3=factory.getBean("s3",String.class);
        System.out.println("s3 data"+s3+" s3 class name"+s3.getClass());
    }
}
```

Singleton java class

- The java class that allows us to create only one object per jvm is called ***singleton***javaclass.
- Take singleton java class in the following situations
  a) If class is not having any state
  b) If class is having only sharable read only state (final variables)
  c) If class is having huge amount of state and performing read, write operations on that state in synchronized manner (1 thread at a time) (implementing caching logic)
- Instead of creating multiple objects with same data or no data, create one object and access it for multiple times for this take the class as singleton javaclass.
- If we just create one object for java class even through class allows us to create multiple objects then that class is not called Singleton java class.

eg: our servlet class is not singleton class because it allows us to create multiple objects but most of the times servlet container creates only 1 object.

Bean Scopes

We can make IOC container keeping our spring bean class objects in different scopes. They are

Spring 1.x

a) singleton
b) prototype

Spring2.x

a) Singleton (default)
b) prototype
c) session
d) request
e) globalSession (removed from Spring 3.x)

Note:globalSessoin is removed in spring 3.x and y.x version
Default scope is "singleton".

Singleton:
Returns same bean class object for all *factory.getBean(-)* method calls with same bean id.

Prototype:
Returns separate bean class object for every *factory.getBean(-)* method call.

Request:
Bean class object is specific to each request.

Session:
Bean class object is visible through out session

GolbalSession:
Specific to portlets env.....
Use "scope" attribute of <bean> tag to specify bean scope.

Note:request, session scopes can be used only in spring MVC applications.
eg:applicationContext.xml

```
<bean id="v1" class="com.nt.beans.vehicle" scope="prototype">
.......................
.....................
</bean>
Vehicle v1=factory.getBean("v1", vehicle.class);
 Vehicle v2=factory.getBean("v1", vehicle.class);
System.out.println (v1.hashcode()+""+v2.hashcode());
System.out.println (v1==v2);
```

Cache:

- Cache is a temporary memory that holds data for temporary period (buffer).if Dao class uses cache support it means database data like countries and uses the across multiple same request and reduces no of hits to Database s/w.
- We generally configure spring bean that is acting as dao class as singleton scope spring bean because it contains SQL queries as the read only state(final static variables).
- The recommendation that are given to take java class as singleton java class can also be taken as recommendation to take spring bean as singleton scope spring bean.

Note:

*The Bean scope '**singleton**' never makes our bean class as singleton java class. But gives singleton behavior while creating object.*

- If Bean scope is singleton then IOC container creates the bean class object and keeps in HashMap element as value by having bean 'id' as key and uses that object across the multiple "*factory.getBean(-)*"method calls.
- If scope is prototype then IOC contains doesn't keep the created bean class object in HashMap so it returns new object for every "*factory.getBean(-)*" call
- If scope is "request" then bean object will be maintained as request attribute.
- If scope is "Session" then the bean object will be maintained as a session attribute.

* We generally take DAO as singletone bean class bcz it contains final read-only declarable data source object.

----------0----------

~~What is bean wiring?~~

~~What is auto wiring?~~ explain diff. ways of creating bean cls obj through IOC container?

1) 0-Param constructor (When no properties are configured for dependency injection and when only setter injection enable)
2) Parameterized constructor(When atleast 1 property is configured for constructor)
3) Instance factory method (When factory-method attribute is specified)
4) Static factory method (when factory-bean attribute is specified)

Bean Wiring

Configuring beans and their dependency injection in spring bean configuration file is called wiring.

1) **Explicit wiring:** Here we use *<property>*, *<constructor-arg>* tags for dependency injection configuration.
2) **Auto Wiring:** Here IOC container automatically detects dependents and injects then to target objects. Here no *<property>*, *<constructor-arg>* are required.

Autowiring/implicit wiring limitations:

- Can be used as only to inject objects but not the simple values.
- If IOC container have multiple dependencies to inject then ambiguity problem may raise.
- Kills the readability of xml file, so the bug fixing becomes very complex.

Note: Use autowiringonly in POC projects or PILOT project for faster/Rapid application development.

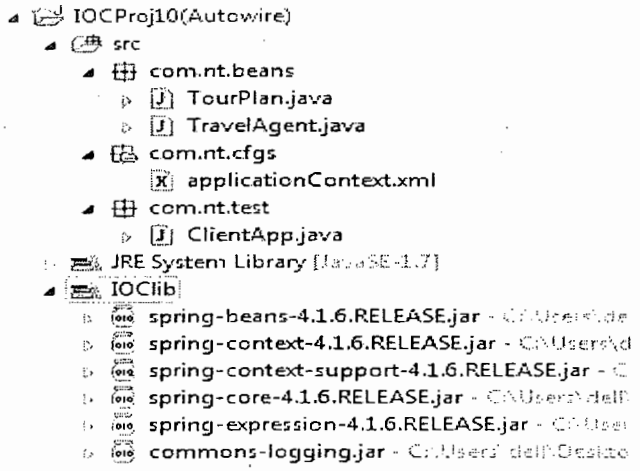
By using "autowire" attribute of <bean> tag are can enable autoworing.

There are 4 modles of performing autowiring

1) byName
2) byType
3) constructor
4) autodetect (removed from spring 3.x)

autowire="byName"

- Performs setter injection
- For all target class property name and dependent class bean id must match.
- Here there is no possibility of getting ambiguity problem




```
//TravelAgent.java
package com.nt.beans;
public class TravelAgent {
        private TourPlan  tp;
         public TravelAgent(){
                System.out.println("TravelAgent:0-param constructor");
          }
         public TravelAgent(TourPlan tp){
                System.out.println("TravelAgent:1-param constructor");
           this.tp=tp;
         }
         public void setTp(TourPlan tp){
                System.out.println("TravelAgent:setTp(-)");
           this.tp=tp;
         }
         // toString()
         @Override
              public String toString() {
                     return "TravelAgent [tp=" + tp + "]";
              }      }
```

TourPlan.java (dependent class)

```
package com.nt.beans;
import java.util.Arrays;
public class TourPlan {
    private String[] places;
     //setter method
     public void setPlaces(String[] places) {
                    this.places = places;
            }
     // toStirng()
    @Override
    public String toString() {
            return "TourPlan [places=" + Arrays.toString(places) + "]";
    }
}
```

applicationContext.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.xsd">
<bean id="tp"  class="com.nt.beans.TourPlan">
<property name="places"><!-- explicit wiring -->
<list>
<value>Delhi</value>
<value>simla</value>
</list>
</property>
</bean>
<bean id="tp2"  class="com.nt.beans.TourPlan">
<property name="places"><!-- explicit wiring -->
<list>
<value>NewYork</value>
<value>Goa</value>
</list>
</property>
</bean>
<bean id="tagent"  class="com.nt.beans.TravelAgent"   autowire="constructor">
<constructor-arg ref="tp2"/>
</bean>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.beans.TravelAgent;
public class ClientApp {
    public static void main(String[] args) {
```


the autowire possible value are ByName ByType Constructor no autoDetect Default(default)

OTE* while cfg bean properties for wiring we can cfg two properties for explicitly wire and some two properties for autowiring.


```
        //Create IOC Container
        BeanFactory factory =new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
        // Get Bean
        TravelAgent agent=(TravelAgent)factory.getBean("tagent");
        System.out.println(agent);
    }//main
}//class
```

autowire="byType":

- perform setter injection
- For this target class property type and dependent class must be match or must be compatable with eachother.
- There is possibility of getting ambiguity problem.

eg:

```
<bean id="tplan" class="com.nt.beans.Tourplan">
.................................
...................
<bean id="target" class="com.nt.beans.TravelAgent" autowiring="byType"/>
```

Note: If we configure "TourPlan" class one more time in spring bean configuration file with different bean id then we get ambiguity problem.

atuowire="constructor"

- uses parameterized constructor to perform constructor injection.
- There is a possibility of getting ambiguity problem.If multiple dependencies are found, but it doesn't throw exception if target class contains 0-param constructor otherwise exception will be thrown.

eg: *<bean id="tp1" class="com.nt.beans.TravelAgent" autowire="constructor">*

autowire="autodetect" *(Try in spring 2.x)*

- If target class contains 0-param constructor then it performs "byType" mode autowiring.
- If target class contains parameterized constructor then it performs "constructor" mode auto wiring.
- Removed from spring 3.0 onwards

```
eg:<bean id="tp1" class="com.nt.beans.TourPlan">
...................
...................
</bean>
<bean id="target" class="com.nt.beans.TravelAgent" autowire="autodetect">
```

autowire ="no"

Disable autowiring, even though default autowire is specified in the beanss tag using default-autowire attribute, The possible values of autowire attribute are no, default, byType, byName, constructor, autodetect.

What happens if we enable both auto wiring and explicit wiring on same property? <beans default-autowire="ByName".....

Ans: If both wirings are performing the same constructor injection or [illegible] that [illegible] value [illegible] wiring [illegible] injections then setter injection will [illegible]

Dependency-Check

While performing constructor injection by using certain parameterized constructor, all the params of that constructor must be configured for wiring, if not configured, exception will be raised.

The above restriction is not there for setter injection configuration.To enable such restriction while working with setter injection go for dependency check by adding "dependency-check" attribute in *<bean>* tag.

dependency-check="simple":

Checks whether all simple properties are configured for setter injection or not, if not configured then exception will be raised.

dependency-Check="objects":

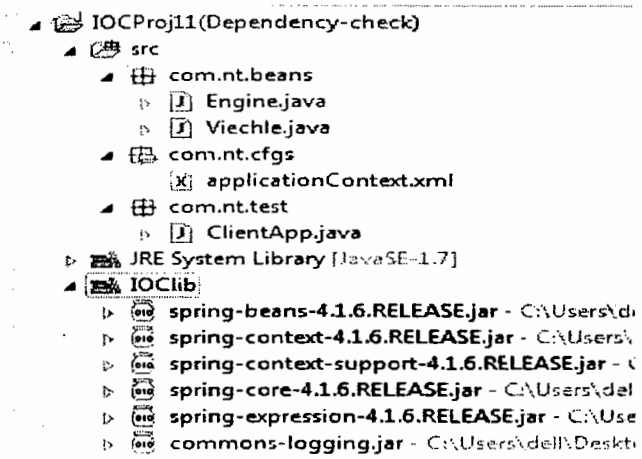
Checks whether all reference type properties are configured for setter injection or not.

dependency-check="all":

Checks whether both simple and object properties are configured for setter injection or not.

dependency-check="none":

Disables dependency check. If no value is given default "none" will taken. We must use Spring-beans-2.0.xsd in *applicationContext.xml.*



eg:

Vehicle.java

```
package com.nt.beans;
public class Viechle {
    private int id;
    private Engine engine;
    public void setId(int id) {
        this.id = id;
    }
    public void setEngine(Engine engine) {
        this.engine = engine;
    }
    @Override
    public String toString() {
        return "Viechle [id=" + id + ", engine=" + engine + "]";
    }
}
```

Engine.java

```
package com.nt.beans;
public class Engine {
    private String type;
    public void setType(String type) {
            this.type = type;
    }
    @Override
    public String toString() {
            return "Engine [type=" + type + "]";
    }
}
```

applicationContext.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.2.0.xsd">
<bean id="eng"  class="com.nt.beans.Engine">
<property name="type"  value="diesel"/>
</bean>
<bean id="viechle" class="com.nt.beans.Viechle" dependency-check="all">
<!-- <property name="id"  value="1010"/> -->
<property name="engine" ref="eng"/>
</bean>
</beans>
```

ClientApp.java:

```
package test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.beans.Viechle;
public class ClientApp {
    public static void main(String[] args) {
            //Create IOC Container
BeanFactory factory =new XmlBeanFactory(new
FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
            //Get Bean
            Viechle v=factory.getBean("viechle",Viechle.class);
            System.out.println(v);
    }//main
}//class
```

Note:"dependency-check" is removed from spring ~~2.5~~ 3.0 because it allow to apply restriction on group of properties not on specified properties. "@Required" is given as an alternate.

we can use following code to crete xml bean factory to create bean factory container. better alternate is working with applicationcontext container

```
//locating Spring cfg file
Resource res=new FileSystemResource("src/main/java/com/nt/cfgs/apllicationContext.xml
//create IOC container
BeanFactory factory=new DefaultListableBeanFactory();
XmlBeanDefinationReader reader=new XmlBeanDefinitionReader(factory);
```



```
reader.loadBeanDefinitions(res);
 //get bean Student st=factory.getBean("st",Student.class);
```

Nested Beanfactory

In real time project contains multiple layers. Instead of configuring all the beans of these multiple layers in single IOC container, it is recommended to take multiple IOC containers on "*one- per-layer*" basis.To use the beans of 1 IOC container in another IOC container it is recommended to make them as Nested containers like parent container and child container. The beans of parent container can be used in child container beans but reverse is not possible.Generally service layer beans want to use the beans of persistence layer, so we configure beans of persistence layer in parent container and the beans of service layer in child container.

Parent IOC container/ (Parent factory)

Manages business tier comps like Service class, DAO class and etc.

Child container/Nestead container/child factory

Manages presentation tier comps like Controller, View.

Note :

1) The components that takes inputs, displaying outputs by handling req, res are called presentation tier components(Controller,View).

2) The components that deals with business logic, persistence logic are called business tier components (Service, DAO classes).

3) It is always recommended to take two different IOC containers as nested/parent, child container to manage ~~persistence~~ presentation tier, business tier components.

4) Since presentation tier components use business tire components manage business tier components in parent IOC container and manage presentation tier components in child IOC container.

- IOCProj12(NestedBeanFactories)
  - src
    - com.nt.beans
      - BankLoanApprover.java
        - BankLoanApprover
      - LoanDetails.java
    - com.nt.cfgs
      - approval-beans.xml
      - loan-beans.xml
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\de
    - spring-context-4.1.6.RELEASE.jar - C:\Users\c
    - spring-context-support-4.1.6.RELEASE.jar - C
    - spring-core-4.1.6.RELEASE.jar - C:\Users\dell
    - spring-expression-4.1.6.RELEASE.jar - C:\User
    - commons-logging.jar - C:\Users\dell\Deskto

LoanDetails.java :

```
package com.nt.beans;
public class LoanDetails {
    private int loanId;
    private String loanType;
    private String customer;
    public LoanDetails(){
            System.out.println("LoanDetails:0-param constructor");
    }
    //setters and getters
    public void setLoanId(int loanId) {
            this.loanId = loanId;
    }
    public void setLoanType(String loanType) {
            this.loanType = loanType;
    }
    public void setCustomer(String customer) {
            this.customer = customer;
    }
    public int getLoanId() {
            return loanId;
    }
    public String getLoanType() {
            return loanType;
    }
    public String getCustomer() {
            return customer;
    }
    //toString()
    @Override
    public String toString() {
            return "LoanDetails [loanId=" + loanId + ", loanType=" + loanType
                    + ", customer=" + customer + "]";
    }
}
```

BankLaonApprover.java

```
package com.nt.beans;
public class BankLoanApprover {
    private LoanDetails details;
    public BankLoanApprover(){
            System.out.println("BankLoanApprover:0-param constructor");
    }
    //setters
    public void setDetails(LoanDetails details) {
            this.details = details;
    }
    // toString()
    @Override
    public String toString() {
```

```
        return "BankLoanApprover [details=" + details + "]";
    }
    public String approveLoan(){
        if(details.getLoanType().equals("two-wheeler") ){
            return "Loan is approved";
        }
        else{
            return "Loan in rejected";
        }    }}
```

loan-beans.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.xsd">
<bean id="ld"  class="com.nt.beans.LoanDetails">
<property name="loanId" value="11"/>
<property name="loanType" value="two-wheeler"/>
<property name="customer" value="raja"/>
</bean>
</beans>
```

approval-beans.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.xsd">

<bean id="ld"  class="com.nt.beans.LoanDetails">
<property name="loanId" value="11"/>
<property name="loanType" value="3-wheeler"/>
<property name="customer" value="raja"/>
</bean>
<bean id="bla" class="com.nt.beans.BankLoanApprover">
<property name="details"  ref="ld"/>(or) <property name = "details" >
                                          <ref local/parent bean = "loanDetails"/>
                                          </property>
</bean>
</beans>
```

ClientApp.java:

```
package test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.beans.BankLoanApprover;
public class ClientApp {
    public static void main(String[] args) {
        //Create parent IOC container (Parent BeanFactory)
        BeanFactory pfactory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/loan-beans.xml"));
        //create child IOC container (child BeanFactory)
```

```
        BeanFactory cfactory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/approval-beans.xml"),pfactory);
        // get Bean
    BankLoanApprover  approver=cfactory.getBean("bla",BankLoanApprover.class);
        System.out.println(approver.approveLoan());
    }//main
}//class
```

While working with nested BeanFactory/NestedContainers the springBean configuration file associated with child container is called child configuration file is called parent configuration file.
,assosiated with parent container

Can you explain various attributes of <ref>tag?

Ans:

1) <ref local="<bean id>">

Locates given bean id based spring bean configuration from current/local spring bean configuration (Child configuration File)

eg:<*property name="details"*>
<*ref local="id"*/>
</*property*>

2)<ref parent="<bean id>"

Locates the given bean id based spring bean configuration from parent spring bean configuration file.

eg:<*property name="details"*>
<*ref parent="id"*/>
</*property*>

3)<ref bean="<beanid>">:

Frist tries to locate from current/local bean configuration file if file not available then it tries to locate from parent bean configuration file

eg:<property name="details">
<ref bean="id"/>
</property>
<property name="details" ref="id"/> is equal to <ref bean="id"/>

P-Name space and C-Name space

In xml schema "name space" is a library that contains set of xml tags. Every name space is identified with its name space url.In spring 3.0 p-name space is introduced to perform setter injection configuration without using lengthy <*property*> tags. It allows to place attribute directly in <*bean*> tags having the following syntax.

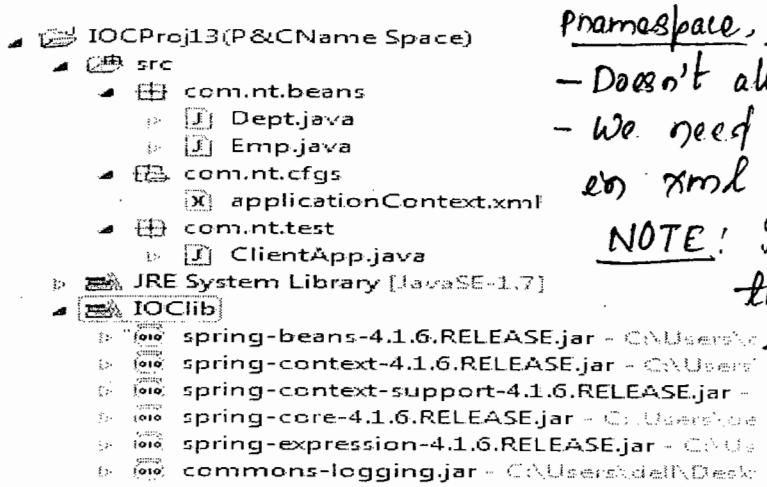
p:<property name>="Value"
p:<property name>-ref="<bean id>"
p-name space url is: http://www.springframework.org/schema/p

Similarly c-name space introduced from spring 3.1.1 version for constructor injection configuration by avoiding the lengthy ***<constructor-arg>*****tags**.It allows to place attributes directly in<*bean*> tag having following syntax.

c:<property name>="value"
c:<property name>-ref="<bean id>"
c-name space uri is:http://www.springfranework.org/schema/c (Collect from spring reference)

Note:

p-namespace and c-namespace allows to writeshorter configurations written in <bean> tag itself to perform setter, constructor injection configuration.



Pnamespace, Cnamespace limitations:
- Doesn't allow to work with Collection injection
- We need to import separate namespace in xml document.

NOTE: Industry is habituated to use the traditional <property> tag, <constructor-arg> tag Setter, Constructor inject.

limitation:-takes time to move and habituate with p,c namespace style of dependency injection configuration
Not came into limelight beacause of anottation arrival to spring programming.

Emp.java

```
package com.nt.beans;
public class Emp {
    private int eno;
    private String ename;
    private Dept dept;
    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public Dept getDept() {
        return dept;
    }
    public void setDept(Dept dept) {
        this.dept = dept;
    }
    @Override
    public String toString() {
        return "Emp [eno=" + eno + ", ename=" + ename + ", dept=" + dept + "]";
    }
}
```

Dept.java

```
package com.nt.beans;
import java.util.Date;
public class Dept {
    private int deptno;
    private String name;
    private Date dop;
    public Dept(int deptno, String name, Date dop) {
            System.out.println("Dept:3-param constructor");
            this.deptno = deptno;
            this.name = name;
            this.dop = dop;
    }
    @Override
    public String toString() {
            return "Dept [deptno=" + deptno + ", name=" + name + ", dop=" + dop
                    + "]";
    }
}
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:c="http://www.springframework.org/schema/c"
 xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="dt" class="java.util.Date" p:year="115"
                                        p:month="10"
                                        p:date="20"/>
<bean id="dept"  class="com.nt.beans.Dept"
                    c:deptno="8001"
                    c:name="Accounts"
                     c:dop-ref="dt"/>
<bean id="emp" class="com.nt.beans.Emp"
                   p:eno="101"
                   p:ename="raja"
                   p:dept-ref="dept"/>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Emp;
public class ClientApp {
    public static void main(String[] args) {
            // create IOC container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //BeanFactory factory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
            //Get Bean
            Emp emp=ctx.getBean("emp",Emp.class);
            System.out.println(emp);
    }//main
}//class
```

Working with ApplicationContext container (IOC container)

- BeanFactory Container means it is the IOC container that is created by creting BeanFactory object
- ApplicationContext Container means it is the IOC container that is created by creating ApplicationContext object.
- ApplicationContext object is the object of java class that implements ***org.springframework.contex.ApplicationContex(I)***(It is the sub interface of *BeanFactory(I)*)
- **There are three popular implementation classes for *ApplicationContext(I)*** using which we can create *ApplicationContext* container (IOC container)

1)FileSystemXmlApplicationContext:

Creates ApplicationContext container by locating spring bean configuration file from the specified path of FileSystem.

eg:

```
ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com.nt/cfgs/applicationContext.xml");
```

2)ClassPathXmlApplication Context:

Create ApplicationContext container by locating spring bean configuration file from the directories and jar files added to classpath.

eg:

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

3) XmlWebApplicationContext:

Creates ApplicationContext container in web applications.

ApplicationContext container can perform all the operations of BeanFactory container and also perform these additional operations:

a) Pre-installation of singleton spring Beans.
b) Ability to work with properties file.
c) Support for i18n (internatiolization).
d) Support for event handling /processing.
e) Ability to start,stop and close container.
f) Automatic registration of post processor classes. And etc.

While working with BeanFactory container the container instantiates spring bean only when *factory.getBean(-)* method is called (for any scope bean).**While working with ApplicationContext container the singleton scope beans will be instantiated the moment the container is started.**This is called Pre-instantiation of singleton Beans.

If prototype scope in dependent to singleton scope then prototype scope bean will also be preinstantiated to support dependency injection on singleton scope bean. But it doesn't mean prototype scope bean has changed scope to "Singleton".Bean pre-instatiation is very much similar to enabiling <load-on-startup> on servlet.

Q)We have spring bean configuration file with 10 beans. How can we enable pre-instantiation only on 6 beans?
Ans: Take 6 beans as singleton scope beans and other 4 beans as "prototype" scope beans and don't make 'prototype' scope beans as dependent to singleton scope beans.

Q)Why there is no pre-instantiation on prototype scope beans while working with applicationcontext container?
Ans: IOC container creates bean class object for every (request) factory.getBean (-,-) method call. So even though object is created through pre-instantiation process that will be wasted because it has to create new object when factory.getBean (-,-) method is called. So there is nopre-instantiation on prototype scope beans.

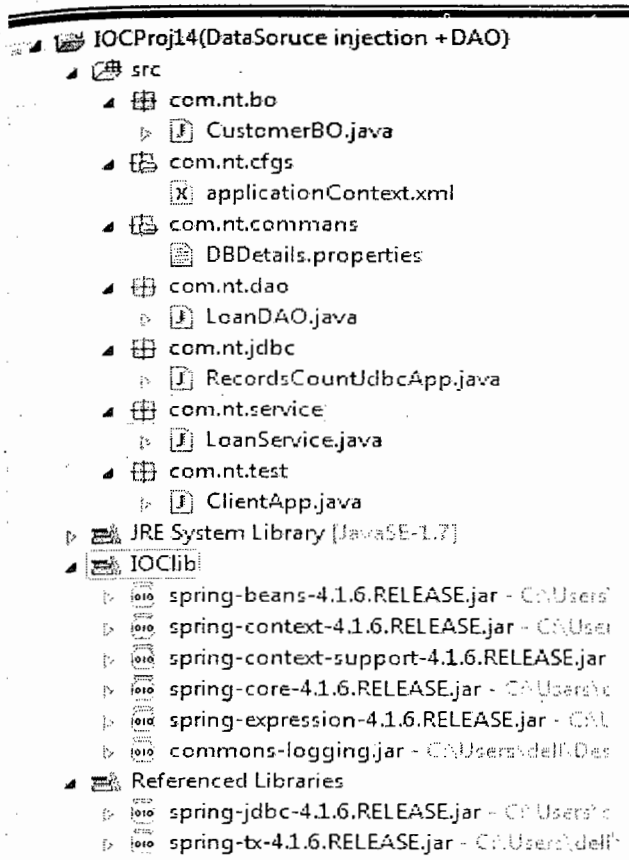
The java class that contains persistence logic is called DAO.The java class that contains business logic is called service/Business service class.Generally service class needs DAO class object.JDBC DataSource object represent JDBC connections pool and we can use this JDBC DataSource object to get connection object from JDBC connection pool.Spring api gives *org.springframework.jdbc.datasource.DriverManagerDataSource* class to give dataSource object pointing to the IOC container managed JDBC connection pool. It creates the connection object in connection pool based on the bean property values configuration. They are driverClassName, url, username, password.

- Factory.getBean(-) method having singleton scope bean id creates and returns bean class object.
- ctx.getBean(-) method having singleton scope bean id returns already pre-instantiated bean class object.

if prototype scope bean is dependent to singleton scope bean then the applicationcontext container also instantiated prototype scope bean along with singleton scope bean that means both beans instantiated through pre-instanstiation process.

bean factory container perform lazy instancetiation of singlton scope beans that mean it can not instantiate bean until factory.getbean method is called.

applicationcontext container perform egar instantiation/ early instantiation of singleton scope bean that means. singleton scope bean instanstiated the movement ioc container is created(applicationcontext).



applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:c="http://www.springframework.org/schema/c"
 xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="ppc"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<property name="location" value="com/nt/commons/DBDetails.properties"/>
</bean>
<bean id="drds"  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="${jdbc.driver}"/>
<property name="url" value="${jdbc.url}"/>
<property name="username" value="${db.user}"/>
<property name="password" value="${db.pwd}"/>
</bean>
<bean id="dao"  class="com.nt.dao.LoanDAO">
<property name="ds"  ref="drds"/>
</bean>
<bean id="service" class="com.nt.service.LoanService">
<property name="dao" ref="dao"/>
</bean>
</beans>
```

to diasable pre-instantiation of singleton scope bean we can be use lazy-init attribue of bean tag

```
<bean id="flipkart"
        class="pkg.Flipkart" scope="singleton" lazy-init="true"/>
```

Q: we have 10 single Scope beans in ApplicationContext container can u tell me how to enable pre-instantiation only 5 beans

A: enable lazy loading on 5 beans and dont make this 5 beans as dependent beans to other 5 singlton scope beans

Customer.java

```
package com.nt.bo;
public class CustomerBO {
    private int cno;
    private String cname;
    private float pamt,intramt;

    public int getCno() {
            return cno;
    }
    public void setCno(int cno) {
            this.cno = cno;
    }
    public String getCname() {
            return cname;
    }
    public void setCname(String cname) {
            this.cname = cname;
    }
    public float getPamt() {
            return pamt;
    }
    public void setPamt(float pamt) {
            this.pamt = pamt;
    }
    public float getIntramt() {
            return intramt;
    }
    public void setIntramt(float intramt) {
            this.intramt = intramt;
    }
}
```

DBDetails.properties

```
# jdbc properties
db.user=scott
db.pwd=tiger
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url =jdbc:oracle:thin:@localhost:1521:xe
```

LoanDAO.java

```
package com.nt.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.sql.DataSource;
import com.nt.bo.CustomerBO;
public class LoanDAO {
```

```
    private static final String INSERT_QRY="INSERT INTO CUSTOMER_LOANINFO
VALUES(?,?,?,?)";
    private  DataSource ds;
  public void setDs(DataSource ds) {
            this.ds = ds;
    }
 public int insert(CustomerBO cbo){
  Connection con=null;
  PreparedStatement ps=null;
  int result=0;
    try{
            // get Connection object from jdbc con pool
            con=ds.getConnection();
            //create PreparedStatement object
            ps=con.prepareStatement(INSERT_QRY);
            //set values to Query param
            ps.setInt(1,cbo.getCno());
            ps.setString(2,cbo.getCname());
            ps.setFloat(3,cbo.getPamt());
            ps.setFloat(4,cbo.getIntramt());
            //execute the Query
            result=ps.executeUpdate();
            }
     catch(SQLException se){
              se.printStackTrace();
     }
    catch(Exception e){
            e.printStackTrace();
    }
    finally{
            //jdbc objects
            try{
                    if(ps!=null)
                            ps.close();
            }
            catch(SQLException se){
                    se.printStackTrace();
            }
            try{
                    if(con!=null)
                            con.close();
            }
            catch(SQLException se){
                    se.printStackTrace();
            }
    }//finally
    return result;
  }//method
 }//class
```

RecordsCountJdbcApp.java

```
package com.nt.jdbc;
import java.io.FileInputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;
public class RecordsCountJdbcApp {
    public static void main(String[] args) throws Exception{
            // locate properties file
            InputStream is=new
FileInputStream("src/com/nt/commons/DBDetails.properties");
            // Load properties info to java.util.Properties class object
            Properties props=new Properties();
            props.load(is);
            Class.forName(props.getProperty("jdbc.driver"));
            Connection con=DriverManager.getConnection(props.getProperty("jdbc.url"),
            props.getProperty("db.user"),
            props.getProperty("db.pwd"));
            Statement st=con.createStatement();
            ResultSet rs=st.executeQuery("select count(*) from student");
            int cnt=0;
            if(rs.next()){
              cnt=rs.getInt(1);
            }
            System.out.println("Records count"+cnt);

            //close jdbc objects
            rs.close();
            st.close();
            con.close();
            }//main
    }//class
```

LoanService.java

```
package com.nt.service;
import com.nt.bo.CustomerBO;
import com.nt.dao.LoanDAO;
public class LoanService {
    private LoanDAO dao;
    public void setDao(LoanDAO dao) {
            this.dao = dao;
    }
    public float  calcIntrAmt(int cno,String cname,float pamt, float rate,float time){
            //write business logic
            float intramt=pamt*rate*time/100.0f;
            //use DAO class peristence logic
```

```
            CustomerBO cbo=new CustomerBO();
            cbo.setPamt(pamt);
            cbo.setCno(cno);
            cbo.setCname(cname);
            cbo.setIntramt(intramt);
            int result=dao.insert(cbo);
            return intramt;
        }//method
    }//class
```

Client.java

```
package test;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.config.BeanFactoryPostProcessor;
import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
import org.springframework.beans.factory.config.PropertyPlaceholderConfigurer;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import com.nt.service.LoanService;
public class ClientApp {
    public static void main(String[] args) {
            // Create IOC container
            //ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
             BeanFactory factory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
             //get  PropertyPlaceholderConfigurer object (pphc)
            BeanFactoryPostProcessor
    bfpp=factory.getBean("ppc",PropertyPlaceholderConfigurer.class);
             bfpp.postProcessBeanFactory((ConfigurableListableBeanFactory)
factory);
            //get Bean
            LoanService service=ctx.getBean("service",LoanService.class);
            // call b.method
            float intramt=service.calcIntrAmt(1009,"Raja",9000,2,20);
            System.out.println("Intr Amount"+intramt);
    } }//class
```

The java class with standards (minimum getter & setter methods) is called javaBean.

In realtime the data between 2 component/resources/layer will not be sent as individual values. They will be placed in java bean class object and will be sent to from source layer to destination layer.

The javaBean whose object holds the end user supplied i/p values (from data) is called VO class (Value Object).

The javaBean whose object holds the data required for persistence logic or the data generated by persistence logic is called Bo class(Business Object). The javaBean whose object holds data to transpiring frameworker from 1 layer to another layer is called DTO class. This class generally implements "Serializable" interface.

VO ---> value object

BO ---->Data Transfer Object

DTO--->DataTranspringframeworker object

Generally the DAO Class contains 2 parts
a) Query Part
b) Code Part

Note:

By using **"locations"** property we can configure multiple properties files in spring applicationcontext containers environment.In this process if all multiple properties files contains same key with different value then the lastly configured file values will be taken by overriding other file values.

eg:

```
<bean id="ppc"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<property name="locations">
<list>
< value>src/com/nt/commons/DBDetails1.properties</value>
< value>src/com/nt/commons/DBDetails2.properties</value>
</list>
</property>
</bean>
```

Working with properties file

The text file that maintains the entries as key value pairs is called properties file.We generally use properties file to maintain jdbc properties like *driverClassName, url, username and password* and make them as flexible to modify without touching the source code.Spring application can achive this flexibility through xml file (spring bean configuration file) If spring application is added as new app to the existing project and if existing applications of that project are getting jdbc properties from properties file then ~~are~~ we should make sure that spring application also get jdbc properties from properties file.

In ApplicationContext container enivnorment we need to configure "*org.springframework.beans.factory.config.PropertyPlaceHolderConfigurer*" class as spring bean to make container tolocate given properties file and to recognize place holders in spring bean configuration file.

Place holders---->${--------}

Each placeholder represents 1 key of the property file to get values from properties file.

DBDetails.properties(com/nt/commons)

```
jdbc.driver = oracle.jdbc.driver.OracleDriver
jdbc.url =jdbc:oracle:thin:@localhost:1521:xe
db.user = system
db.pwd = manager
```

applicationContext.xml

```
<bean id="ppc"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="src/com/nt/commons/DBDetails.properties"/>
</bean>
<bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.user}"/>
        <property name="password" value="${jdbc.pwd}"/>
</bean>
```

I18n (Internationalization)

- **Locale means country+language**

eg: en-us
en-Br (English as it speaks in Britan)
fr-FR
hi-IN

- Making our application working for different "locales" is called enabling I18n (internalization). It deals with displaying presentation labels.
- Like Displaying numbers,currency symbols, displaying Date values
- In core java, JEE application we use ResourceBundle, Locale classes of java.util.package to work with I18n.
- To apply I18n we need to take multiple properties file having presentation labels on 1 per "locale" basis. All these files must have same keys and different values, we can get these values of different languages by using **google translator.**

eg: App.properties (Base file)
App_de_DE.properties (for German)
And etc....

- If no mataning file is found the base file will be automatically used.

We can use google translator labels in other language based on the given English word. We can also place Unicode numbers in the properties files to render non English character labels. For that we need to use third party tools like Unicode editor and jdks "native2ascii" tool.

Procedure-1:
1) Download Unicode editor from www.higopi.com and type hindi words (use index.html)
2) Copy hindi words from Unicode editors and place them in notepad.
(save the file by taking "unicode" as encoding type)
Input.txt:
Write hindi words.
3) Use "native2Ascii" tool to get Unicode numbers.
cmd

```
>native2Ascii encoding=Unicode inputs.txt output.txt (contains Unicode numbers)
```

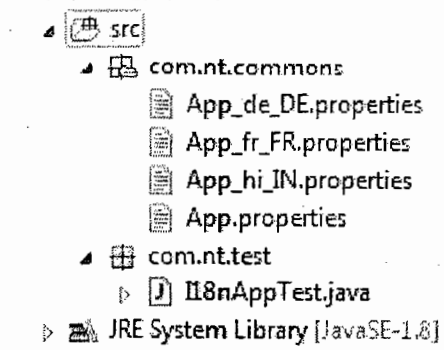
Limitations of JEE,JSE envinorment while working with I18n:

In a web application if multiple Servlet/jsp program wants to display "labels" collected from properties file then we need to create "ResourceBundle" class object in multiple servlet/jsp programs or we need to place "ResourceBundle" object as request attribute value.

Note: *Some plan has to be decided to use "ResourceBundle" object.*

If we want to display values of same key belonging to multiple locale properties file them we need to create multiple "ResourceBundle" class objects.We can overcome these problems in spring envinorment by using "ApplicationContext" container with the support of "ResourceBundleMessageSource" Class.

Directory structure to create plain I18n Application:



I18n in spring

- **Only "ApplicationContext" container supports I18n.**
- We can use "**ResourceBundleMessageSource**" class as spring Bean by specifying base property file name as the value of "baseName" property.
- Since we configure the above class as "singleton scope" SpringBean that the container will creates one object and uses it for multiple times. But the above bean must be configure with fixed bean id "***messageSource***".
- We can call "*ctx.getMessage()*" toread message from properties file by supplying appropriate "Locale" object.
- This "*ctx.getMessage()*" method make container to locate properties file based on locale object data with support of bean whose id="messageSource".

Advantage:

- Just by creating one object for "ResourceBundleMessageSource" we can use properties file in multiple locations.
- Allows to read same message in different languages with one object of above class.

Example:

- IOCProj15(I18n -Event handling)
  - src
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.commans
      - IOCPMListener.java
      - App_de_DE.properites
      - App_fr_CA.properites
      - App_hi_IN.properites
      - App.properties
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\d
    - spring-context-4.1.6.RELEASE.jar - C:\Users\
    - spring-context-support-4.1.6.RELEASE.jar - (
    - spring-core-4.1.6.RELEASE.jar - C:\Users\del
    - spring-expression-4.1.6.RELEASE.jar - C:\Use
    - commons-logging.jar - C:\Users\dell\Desk
  - Referenced Libraries

App.properties

#(BaseFile-->English)
str1=delete
str2=save
str3=stop
str4=cancel

App_hi_IN.properties

str1=\u0928\u0916\u093e\u094a
str2=\u0938\u0941\u0930\u094d\u0915\u093f\u0925\u094d
str3=\u0930\u094a\u0915\u094a
str4=\u0930\u0926\u094d\u0926\u094d

App_de_DE.properties

```
#for german language
str1=LOSCHEN
str2=DENN
str3=ZUG
str4=ABSAGEN
```

App_fr_CA.properties

```
#for French language
str1=EFFACER {0} {1}
str2=SAUF
str3=ARRET
str4=ANNULER
```

ClientApp.java

```
package test;
import java.awt.Container;
import java.awt.FlowLayout;
import java.util.Locale;
import javax.swing.JButton;
import javax.swing.JFrame;
import org.springframework.context.support.fileSystemXmlApplicationContext;

public class ClientApp{
public static void main(String[] args){
//create IOC container
FileSystemXmlApplicationContext ctx= new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
//get Messages
Locale l = new Locale("fr","CA");
String s1=ctx.getMessage("str1",null,"msg1",l);
String s2=ctx.getMessage("str2",null,"msg2",l);
String s3=ctx.getMessage("str3",null,"msg3",l);
String s4=ctx.getMessage("str4",null,"msg4",l);

JFrame jf = new JFrame();
Container cp = jf.getContentPane();

//create btns by getting labels from activated properties file..
JButton b1 = new JButton(s1);
JButton b2 = new JButton(s2);
JButton b3 = new JButton(s3);
JButton b4 = new JButton(s4);

cp.setLayout(new FlowLayout());
cp.add(b1);
cp.add(b2);
cp.add(b3);
cp.add(b4);
```

```
jf.pack();
f.setVisible(true);
ctx.close();
        }
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:c="http://www.springframework.org/schema/c"
 xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="messageSource"
 class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="baseName" value="src/com/nt/commons/App"/>
</bean>
</beans>
```

Spring In8 internally uses one or more objects of ResourceBundle class through ResourceBundleMessageSource class object based on the need but it provide abstraction layer on the process of creating and managing ResourceBundle class object due to this programmer feel easy and comfortable by taking single ResourceBundleMessageSource to enabling I18 .

Q: why should give fixed bean ID "messageSource" for ResourceBundleMessageSource while working with I18

A: IOC container internally uses ResourceBundleMessageSource class object for multiple times to get multiple messages from properties file when ctx.getMessage(----) methods are called. In this process IOC container identified this class with fixed bean ID messageSource,since IOC container code readialy code we can't make IOC container looking another bean id base ResourceBundleMessageSource so we must give that fixed bean id.

Q: what is the diff b/w getMessage(-) and getBean(-) method.

A;-

1- Invakable on both BeanFactory and ApplicationContext	2-- Invokable only ApplicationContext
2-Given to get bean class obj from IOC Container	2- Given message from properties file while working with I18n

Event Handling

Event is an object/action that is raised on the component/object.Event handling means executing logic when event is raised for this we need the support of"EventListners".There are four actors in Event processing/handling.

a)Source object (eg: Button)
b)Event Class (eg: ActionEvent)
c)EventListner (eg: ActionListner)
d)Eventhandling method (eg: *actionPerfomance(-)*)

We can perform "Event handling" on "ApplicationContext" container to keep track of when container is started and stopped. This allows programmer to evaluate the performance of spring code/logic.

In spring environment for event handling

Source object: ApplicationContext container
Event class: ApplicationEvent
Event Listner: ApplicationListner
Event handling methods:*onApplicationEvent(-)* //executes when container startedor stopped.

eg-app:

1)Keep any application ready having the support of "ApplicationContext" container.
2)Develop the javaclass for event handling
3)Configure the above "Listner" class in "applicationContext.xml"
<bean class="com.nt.event.IOCAMOListner"/>
4)Create "ApplicationContext" container and stop that in the client application.

- While executing this application also add **spring-aop-<ver>.jar** file.

IOCPOMListner.java:

```
package com.nt.event;
public class IOCPOMListner implements ApplicationListner{
private Long end;
private Long start;
public void onApplicaitonEvent (ApplicationEvent ae){
If (ae.toString().indexof("Refreshed")!=-1){
start=System.currentTimeMillies();
}
else
end=System.currentTimeMillies();
System.out.println("Container acting time" +(end-start) +"ms");
}
}}
```

ClientApp.java:

```
FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext
("src/com/nt/cfgs/applicationContext.xml");
.....
.....
ctx.close(); //close the container
```

Event handling on IOC container is useful to evaluate and optimize the performance of IOC container.

Note1: *ctx.getBean(-)/factory.getBean(-)* gives Bean class object based on given bean id.

Note2: *ctx.getMessage(-,-)* gives "Locale"specific message collected from properties based on given "Key".

Where to use BeanFactory? Where to use ApplicationContext?

Ans: If we are developing **small scale applications** likes mobile applications, EmbededSystem Applications having the support of spring then **use BeanFactory** (Here few extra KBs are also matters towards application size)

If we are developing **enterprise applications** (like web applications, Distributed applications) then **use ApplicationContext** (always prefer working with ApplicationContext).

Similarities and difference between bean BeanFactory and ApplicationContext container?

Operation	**BeanFactory container**	**ApplicationContext**
Bean instantiation	*Yes*	*Yes*
Bean wiring	*Yes*	*Yes*
Auto Wiring	*Yes*	*Yes*
Properties File Support ***(not directly).***	*Yes*	*Yes*
I18n	*No*	*Yes*
Eventhandling -Eventpublication.	*No*	*Yes*
Automatic Registration of BeanPostProcessor	*No*	*Yes*
Pre-instantiation	no	Yes
Automatic Registration of BeanFactoryPostPreprocessor	*no/~~yes~~*	*Yes*
Nested Container	Yes	Yes
Stop/closing Container	*No*	*Yes*

Note:

Constructor injection, dependency-check or @required keeps the restriction of configuring properties for dependency injection but they don't verify correctness of that data. For this we need to take the support of spring bean life cycle.

Bean Life Cycle

It talks about spring bean Class object creation to destruction.In servlet lifecycle we use following lifecycle method

a)init(-)
b)service (-,-)
c)destroy ()

In ordinary class we use constructor, *finalize()* methods to place initialization and uninitialization logic.
SpringBean allows two lifecycle methods.

a)init() method:
Executes after all dependencies are injected. So it can be used to verify whether injections are done with properly or not (with appropriate value or not).

b)destroy() Method:
Executes where IOC container is about to destroy our bean class object. Useful to place uninitialization logics.
(Where these methods can be costum name)

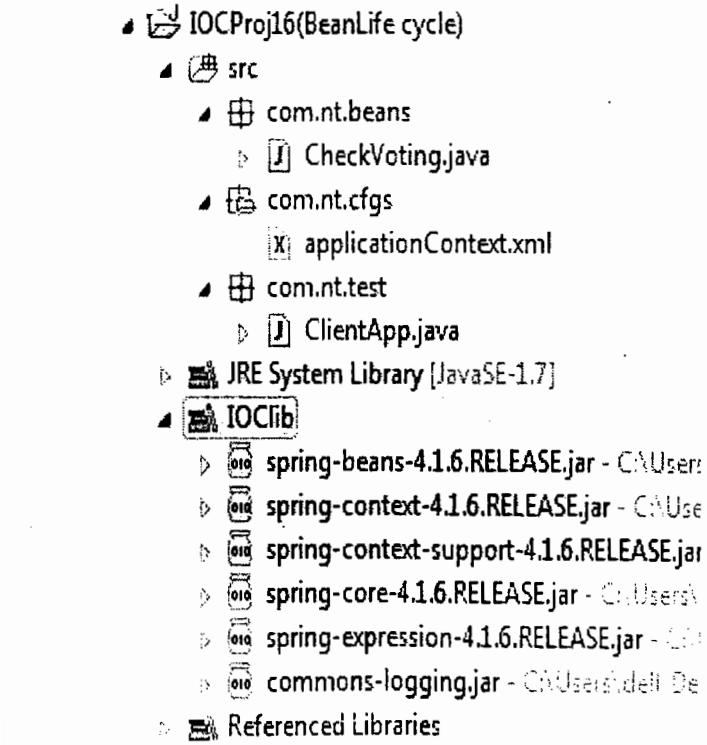
In spring we can perform spring lifecycle operations in three ways:

1) Declarative Approach
2) Programming Approach
3) Annotation Approach

Declarative Approach (xml):

Allows to configure two user-defined methods as bean lifecycle methods. These methods are called custom init methods, custom destroy methods. These methods returntype should be "void" and must not contain "parameters".We must configure the above methods as lifecycle methods using "init-method", "destroy-method" attributes of *<bean>* tag.

Example:



applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:c="http://www.springframework.org/schema/c"
 xmlns:p="http://www.springframework.org/schema/p"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd">
 <bean id="cv" class="com.nt.beans.CheckVoting" init-method="check"
 destroy-method="myDestroy">
 <property name="name"  value="ravi"/>
 <property name="addrs"  value="hyd"/>
 <property name="age"  value="30"/>
 </bean>
 </beans>
```

CheckVoating.java:

```
package com.nt.beans;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
public class CheckVoting implements InitializingBean,DisposableBean {
    private String name,addrs;
    private int age;
    public void setName(String name) {
    System.out.println("setName(-)");
            this.name = name;
    }
    public void setAddrs(String addrs) {
            System.out.println("setAdds(-)");
            this.addrs = addrs;
    }
    public void setAge(int age) {
            System.out.println("setAge(-)");
            this.age = age;
    }
    public void  check(){
            System.out.println("Bean:custom-init()");
            if(name==null || age<=0)
                    throw new IllegalArgumentException("name, age must set with values");
    }
    public  String  checkVotingElgibility(){
            if(age>=18){
                    return name+"  u r  elgible to vote ";
            }
            else{
                    return name+" u r not elgible vote";
            }
    }
```

```
        @Override
        public void destroy() throws Exception {
                System.out.println("Bean:destroy()");
                name=null;
                age=0;
                addrs=null;
        }
        @Override
        public void afterPropertiesSet() throws Exception {
         System.out.println("Bean:AfterPropertiesSet()");
                if(name==null || age<=0)
                        throw new IllegalArgumentException("name, age must set with values");
        }
        public void myDestroy(){
                System.out.println("Bean:custom Destroy()");
                name=null;
                age=0;
                addrs=null;
        }
    }
```

Client.java:

```
    package test;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.support.FileSystemXmlApplicationContext;
    import com.nt.beans.CheckVoting;
    public class ClientApp {
        public static void main(String[] args) {
                // Create IOC container
                ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //get Bean
                CheckVoting cv=(CheckVoting)ctx.getBean("cv");
                System.out.println(cv.checkVotingElgibility());
                ((FileSystemXmlApplicationContext)ctx).close();
        /*      BeanFactory factory=new XmlBeanFactory(new
        FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
                CheckVoting cv=(CheckVoting)factory.getBean("cv");
                System.out.println(cv.checkVotingElgibility());
                ((XmlBeanFactory)factory).destroySingletons(); */
        }
    }
```

In custom "init()" method we generally don't write initialization logic because it won't execute after Bean instantiation it executes after compiling all injection.While woking with "ApplicationContext" container when we close the container by calling "ctx.close()". The container destroys bean class objects when this task is about to happen custom "destroy()" method will execute while working with "BeanFactory" container we can use "*factory.destroySingletons()*" method to destroy all singleton scope bean class objects.

Limitations of Declarative approach of bean Lifecycle:

1)If you forgot to configure lifecycle method in xml file, they will not be executed.

2)Identifying lifecycle method is very complex while dealing with predefined, third party supplied classes as spring beans.

2) <u>Bean LifeCycle programmatic approach:</u>

- Here our spring bean class must implement "InitializationBean(I)" and "DisposibleBean(I)" given by spring api (Bean class becomes api dependent ----->(invasive))
- InitializationBean(I) gives *afterpropertySet()*,DisposibleBean(I) gives *destroy()* method.
- We use "*afterPropertySet()*" method as alternative to custom *init()* and *destroy()* as custom destroy method.

<u>Sample Code:</u>

```
public class checkVoting implements InitializingBean, DisposableBean
{
private int age;
private String name, address;
//setters
..............
...............
public void afterPropertySet()
{
If (age<=0 || name==null)
{
Throw new IllegalArgumentException("name age must be set");
}
public void destroy (){
age=0;
name=null;
address=null;
}
.........
................
}
```

LIMITATION:

bean class invasive class beacause it is implementing spring API interfaces

Note:

There is no need of configure any life cycle methods in xml spring bean configuration.

If we place Bean LifeCycle configuration in both programmatic approach and declarative approach then what happen?

Ans: Programmatic approach configurations execute before declarative approach configurations.

Note1: It is recommended to use 1 approach of bean lifecycle at a time.

Note:2 Programmatic approach makes our springBean class as spring api dependent most of spring api supplied class are given based on programmatic approach of bean lifecycle.

Note3: Custom *init()* and InitializationBean's *afterPropertySet()* is useful to checkwhether important bean properties are injected with values or not.Custom *destroy()*, DisposableBean's *destroy()* is useful to nullify bean properties or to releases non-java resources that are associated with Bean.

3) Bean lifecycle Annotation Approach:

Voter.java

```
package com.nt.beans;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.inject.Named;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Scope;
@Named("vt")
@Scope("singleton")
public class Voter {
@Value("raja")
private String name;
@Value("30")
private int age;
@PostConstruct
public void myInit(){
System.out.println("Voter:myInit()");
if(name==null || age<=0){
throw new IllegalArgumentException(" name,age must be set");
}
}
  @PreDestroy
 public void myDestroy(){
  System.out.println("Voter:myDestroy()");
   name=null;
   age=0;
 }
  public String checkVotingElgibility(){
   if(age<18)
return name+" u r not elgible to vote";
   else
return name+"u  r elgible to vote";
  }
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<context:component-scan base-package="com.nt.beans"/>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Voter;
public class ClientApp {
    public static void main(String[] args) {
            // create IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //get Bean
            Voter vt=ctx.getBean("vt",Voter.class);
            String status=vt.checkVotingElgibility();
            System.out.println("Result::"+status);
            ((FileSystemXmlApplicationContext)ctx).close();
    }
}
```

Conclusion on Bean LifeCycle:

1)While working with user-defined spring bean prefer declarative or annotation approach.
2)While working with third party API supplied classes as spring bean use Declarative approach.
3)While working with spring api supplied as spring bean 1st check for programmatic approach implementation, if not found ~~the~~ then go for declarative approach.

If underlaying container assigns or injects objects/value to target class only when target class implementa/extends certain interface/class is called contextual dependency lookup. Servlet container assisgns config object to our servlet class only when our servlet class implements *javax.servlet.Servlet(I)* directly or indirectly (Contextual dependency lookup).

If one class needs another class then we go for either dependency lookup or dependency injection.If target classs wants to use dependent objects in all the methods then go for dependency injection. If target class wants to use dependent object in specific methods then go for dependency lookup.

If we perform dependency-lookup in regular fashion we need to create one more IOC container in target class to get dependent class object, so instead of creating separate IOC container in target class we ask underlying IOC container ("Client Application started") to inject "BeanFactory or ApplicationContext" object by implementing "BeanFactoryAware or ApplicationContextAware(I)" on target class and we can use that injected objects to get dependent class object.

Here underlying container is injecting object/value **only when target class implemtents "xxxAware" interface**. So it is called Awareinjection or interface injection or contextualdependency lookup (CDL).

Note:

- Aware injection also support to perform dependency lookup to get same dependency class objects in specific methods of target class but it doesn't use separate IOC container in these methods. It uses the underlying IOC container where target bean is running.
- To perform setter and constructor injection IOC container doesn't checks for any contract/condition supported by the beans class. But to perform aware injection the IOC container checks for contract/condition supported by the bean class is implementing xxxAware interfaces or not.

xxxAware interface are:

BeanNameAware----->To inject currentbeanid
BeanFactoryAware------>To inject BeanFactory object
ApplicationContextualAware----->To inject ApplicationContext object.

- IOCProj17(Aware Injection or InterfaceInjection)
  - src
    - com.nt.beans
      - Engine.java
      - Viechle.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\d
    - spring-context-4.1.6.RELEASE.jar - C:\Users\
    - spring-context-support-4.1.6.RELEASE.jar - C
    - spring-core-4.1.6.RELEASE.jar - C:\Users\dell
    - spring-expression-4.1.6.RELEASE.jar - C:\Use
    - commons-logging.jar - C:\Users\dell\Deskt

applicationContext.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.xsd">
<bean id="engg1" class="com.nt.beans.Engine"/>
<bean id="viechle" class="com.nt.beans.Viechle" >
<property name="enggId">
<idref bean="engg1"/>
</property>
</bean>
</beans>
```

Viechle.java

```
package com.nt.beans;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.BeanFactoryAware;
import org.springframework.beans.factory.BeanNameAware;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
public class Viechle  implements ApplicationContextAware{
private  String enggId;
 private ApplicationContext ctx;
  public void setApplicationContext(ApplicationContext ctx)
            throws BeansException {
     System.out.println("Viechle:setApplicationContext(-)");
    this.ctx =ctx ;
 }
public Viechle(){
     System.out.println("Viechle:0-param constructor");
 }
public void setEnggId(String enggId){
    System.out.println("Viechle:setEnggId");
  this.enggId=enggId;
 }
 public void move(){
   Engine engine=ctx.getBean(enggId,Engine.class);
   engine.start();
   System.out.println("Viechle  moved for journey");
  }
}
```

Engine.java

```
package com.nt.beans;
public class Engine{
     public Engine(){
            System.out.println("Engine:0-param constructor");
     }
 public void start(){
    System.out.println("Engg started");
  }
}
```

Client.jav

```
package t
import or springframework.context.ApplicationContext;
import or springframework.context.support.FileSystemXmlApplicationContext;
import co n.nt.beans.Viechle;
public cl s ClientApp {
        p blic static void main(String[] args) {
                ApplicationContext ctx =new
FileSys mXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                Viechle viechle=ctx.getBean("viechle",Viechle.class);
                viechle.move();

}
```

Note:

1)The IOC container that is created by creating "BeanFactory" object. Internally maintain bean id's BeanFactory object.

2)The IOC container that created by creating ApplicationContext object internally maintains bean ids, BeanFactory object, ApplicationContext objects and etc.. All these objects can be injected to any class through Aware injection. This injection can't be used normal and regular values to spring bean. It is useful to inject underlying IOC container maintained objects to spring bean.

FactoryBean

The class that is capable of creating and returning objects is called "Factory" class. **In spring FactoryBean is "selfless" bean i.e it doesn't gives its own class object, it always gives one resultant object.** To develop class as "FactoryBean" class the class must implement the spring API's "FactoryBean(I)" and should provide definition for three methods.

a) *getObject()*---->create and returns resultant object.

b)*getObjectTye()*------>returns java.lang.Class object holding resultant object class name.

c)*IsSingleton()*------>Specifies weather resultant object should be taken as singleton(True) object or prototype object (false)

The Bean that gets objects from jndi registry based on given jndi name will be developed as Factory Bean.

FactoryBean is very important to gather and inject external services objects to target bean class properties. The external services are those services which are not part of our project should be gathered throughout side the project by doing some lookup operations over the network.

- IOCProj18(FactoryBean)
  - src
    - com.nt.beans
      - DateFactoryBean.java
      - ScheduleReminder.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\de
    - spring-context-4.1.6.RELEASE.ja C:\Users\
    - spring-context-support-4.1.6 RELEASE.jar - C
    - spring-core-4.1.6.RELEASE.ja Users\dell
    - spring-expression-4.1.6.RELEASE.jar - C:\Use
    - commons-logging.jar - C:\ dell\Deskto

DateFactoryBean.java

```
package com.nt.beans;
import java.util.Date;
import org.springframework.beans.factory.FactoryBean;
public class DateFactoryBean implements FactoryBean<Date>{
    private int year;
    private int month;
    private int date;
    public DateFactoryBean(int year, int month, int date) {
      System.out.println("DateFactoryBean:3-param constructor");
            this.year = year;
            this.month = month;
            this.date = date;
```

```
    }
    //@Override
    public Date getObject() throws Exception {
            System.out.println("DFB:getObject()");
            return new Date(year,month,date);
    }

    //@Override
    public Class<?> getObjectType() {
            System.out.println("DFB:getObjectType()");
        return Date.class;
    }

    //@Override
    public boolean isSingleton() {
            System.out.println("DFB:isSingleton()");
            return true;
    }}
```

ScheduleReminder.java

```
package com.nt.beans;
import java.util.Date;
public class ScheduleReminder {
    private Date date;
      public void setDate(Date date) {
            this.date = date;
    }
    @Override
    public String toString() {
            return "ScheduleReminder [date=" + date + "]";
    }
}
```

applicationContext.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.xsd">
<bean id="dfb" class="com.nt.beans.DateFactoryBean">
<constructor-arg value="110"/>
<constructor-arg value="10"/>
<constructor-arg value="20"/>
</bean>
<bean id="reminder" class="com.nt.beans.ScheduleReminder">
<property  name="date" ref="dfb"/>
</bean>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.ScheduleReminder;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            // get Bean
            ScheduleReminder reminder=ctx.getBean("reminder",ScheduleReminder.class);
            System.out.println(reminder);
    }
}
```

ServiceLocater(use case of FactoryBean):

The services that are there with in the project are called local services. The services that are external to project but used in the project are called external services. Crickbuzz.com uses ICC score card component as external service. Sharecon.com uses BSE_NSE stock trade component as external service. These external services generally developed by using distributed technologies, ESB, RMI, WebSevice and etc... they will be registered in JNDI registry to expose them for out siders.

The local service class/controller components maintaining JNDI lookup code to get external service component reference from the JNDI registry. But placing JNDI lookup code directly in controller/services components is bad practice and having following limitations.

Note : for problem digram refer page no 103

Limitations:

1)JNDI lookup code will not be reusable code if you write this code in multiple controller/services components.

2) Any change in external service technology or location or JNDI registry details we need to modify the JNDI lookup code in multiple places (service/controller).

To overcome this problems it is recommended to implement a design pattern called ServiceLocator.

- It is a java class having JNDI lookup code to get external service component reference giving reusability for JNDI lookup code and caching support to reduce N/w round trips between application and JNDI registry of external service and also gives flexibility to modify JNDI lookup code without disturbing other components of the application when external service related technology/location/JNDI registry details are changed.
- If ServiceLocator developed as spring bean it recommended to develop it as FactoryBean because though it is depended to other controller/service components of the application those components are not interested in ServiceLocator object there are interested to get the external service components reference that is gathered through JNDI lookup as resultant object.
- If web application uses spring libraries they must be added to build path of the project and WEB-INF/lib folder.
- When we use spring in web application we can't do dependent injection of servlet

component because servlet container manages servlet component and the beans managed by IOC container can't be injected.

- If you want to perform dependency management between two classes both the classes must be managed by IOC container.
- In the above situation servlet component can get spring bean through dependency lookup(ctx.getBean(-,-)).
- When we take ServiceLocator as FactoryBean it actually fulfills the requirement but kills non-invasive principle of spring because we need to implement FactoryBean (I) on ServiceLocator class directly. To overcome this problem we can go for FactoryMethod based bean instantiation as shown below.

ScoreCompServiceLocator.java

```
public class ScoreCompServiceLocator{
private static Map<String,ScoreComp> cache=new HashMap<String,ScoreComp>();
public static ScoreComp getService(){
//actually we need JNDI lookup code here to gather external component reference
If(!cache.containsKey("icc")){
        Cache.put("icc", new ScoreCompImpl());
      }
return cache.get("icc");
   }
}//class
```

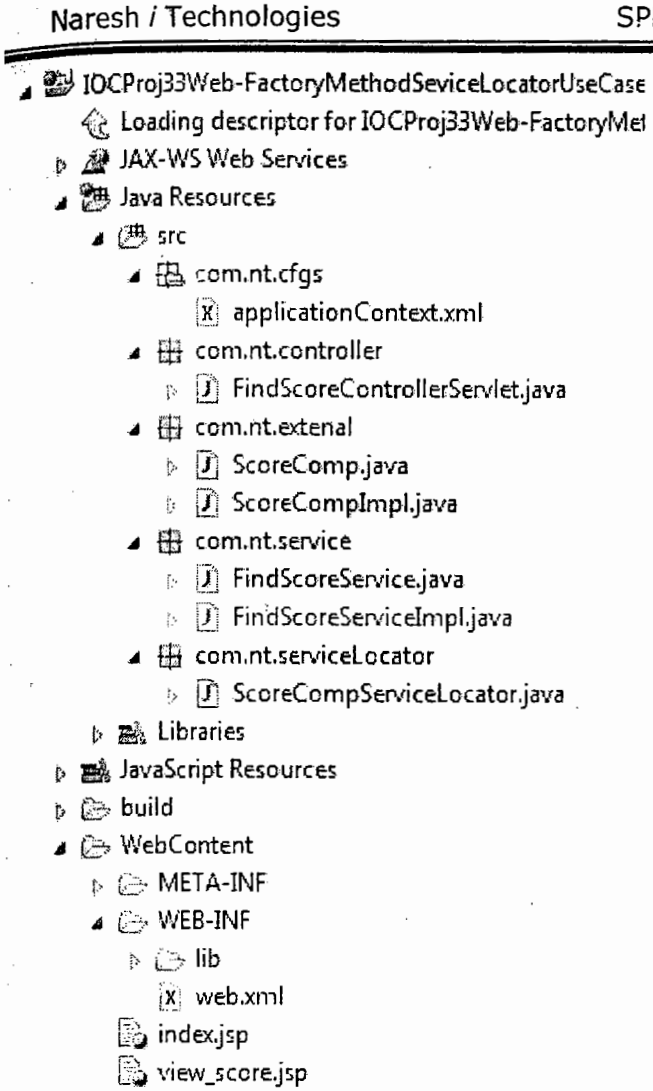
applicationContext.xml:

```
<bean id="serviceLocator" class='com.nt.servicelocator.ScoreCompServiceLocator"
                                factory-method="getService"/>
<bean id="scoreService" class='com.nt.service.FindScoreServiceImpl">
<property name="externalComp" ref="serviceLocator"/>
</bean>
```

How FactoryBean is able to give another object (Resultant Object)?

IOC Container creates "FactortyBean" class object normally based on configuration done in spring bean configuration file, but before keeping that object in a scope, IOC container checks whether the class of the object is implementing "FactoryBean(I)/not" to get Resultant object and calls "isSingleton()" to keep Resultant object as "singleton scope or prototype scope" object with the bean id given for FactoryBean class configuration(Xmlfile).

eg:



applicationContext.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.xsd">

<!-- static factory method bean instantation
<bean id="serviceLocator"  class="com.nt.serviceLocator.ScoreCompServiceLocator" factory-
method="getService"/>
<bean id="scoreService"  class="com.nt.service.FindScoreServiceImpl">
<property name="extenalComp" ref="serviceLocator"/>
</bean> -->

<!-- instance factory method bean instantiation -->
<bean id="serviceLocator"  class="com.nt.serviceLocator.ScoreCompServiceLocator"/>

<bean id="extenalService"  factory-bean="serviceLocator"  factory-method="getService"/>

<bean id="scoreService"  class="com.nt.service.FindScoreServiceImpl">
<property name="extenalComp" ref="extenalService"/>
</bean>
</beans>
```

FindScoreControllerServlet.java

```
package com.nt.controller;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

import com.nt.service.FindScoreService;

publicclass FindScoreControllerServlet extends HttpServlet {
	ApplicationContext ctx=null;

	publicvoid init(){
		//create IOC container
		ctx=new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
	}
	@Override
	publicvoid doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
		int mid;
		FindScoreService service=null;
		String scoreCard=null;
		RequestDispatcher rd=null;
		//read matchId
		mid=Integer.parseInt(req.getParameter("matchId"));
		//get Local Service class objref
		service=ctx.getBean("scoreService",FindScoreService.class);
		//use Service class
		scoreCard=service.getScore(mid);
		//keep the results  in requestScope
		req.setAttribute("scoreCard",scoreCard);
		//forward request to result.jsp
		rd=req.getRequestDispatcher("/view_score.jsp");
		rd.forward(req,res);

	}//doGet(-,-)

}
```

ScoreComp.java

```
package com.nt.extenal;
publicinterface ScoreComp {

        public String findScore(int matchId);

}
```

ScoreCompImpl.java

```
package com.nt.extenal;

publicclass ScoreCompImpl implements ScoreComp {

        @Override
        publicString findScore(int matchId) {
        if(matchId==345)
                return "TestMatch:345/6";
        elseif(matchId==567)
                return "OneDay: 234/9";
        else
                return "Wrong MatchId";
          }//method
        }//class
```

FindScoreService.java

```
package com.nt.service;

publicinterface FindScoreService {
public String  getScore(int matchId);
}
```

FindScoreServiceImpl.java

```
package com.nt.service;

import com.nt.extenal.ScoreComp;

publicclass FindScoreServiceImpl implements FindScoreService {
private ScoreComp  extenalComp;

        publicvoid setExtenalComp(ScoreComp extenalComp) {
        this.extenalComp = extenalComp;
}
        @Override
        public String getScore(int matchId) {
          String scoreCard=null;
           scoreCard=extenalComp.findScore(matchId);
                return scoreCard;
        }
}
```

ScoreCompServiceLocator.java

serviceLocator class with static factory method,instance factory method

```
package com.nt.serviceLocator;

import java.util.HashMap;
import java.util.Map;
import com.nt.extenal.ScoreComp;
import com.nt.extenal.ScoreCompImpl;

publicclass ScoreCompServiceLocator{
	privatestatic Map<String,ScoreComp>cache=new HashMap<String,ScoreComp>();

	/*public static  ScoreComp getService(){
		//actually we need jndi lookup code here to gather extenalcompref
		if(!cache.containsKey("icc")){
			cache.put("icc",new ScoreCompImpl());
		}
		return cache.get("icc");
	}*/

	public  ScoreComp getService(){
		//actually we need jndi lookup code here to gather extenalcompref
		if(!cache.containsKey("icc")){
			cache.put("icc",new ScoreCompImpl());
		}
		returncache.get("icc");
	}

}
```

Web.xml

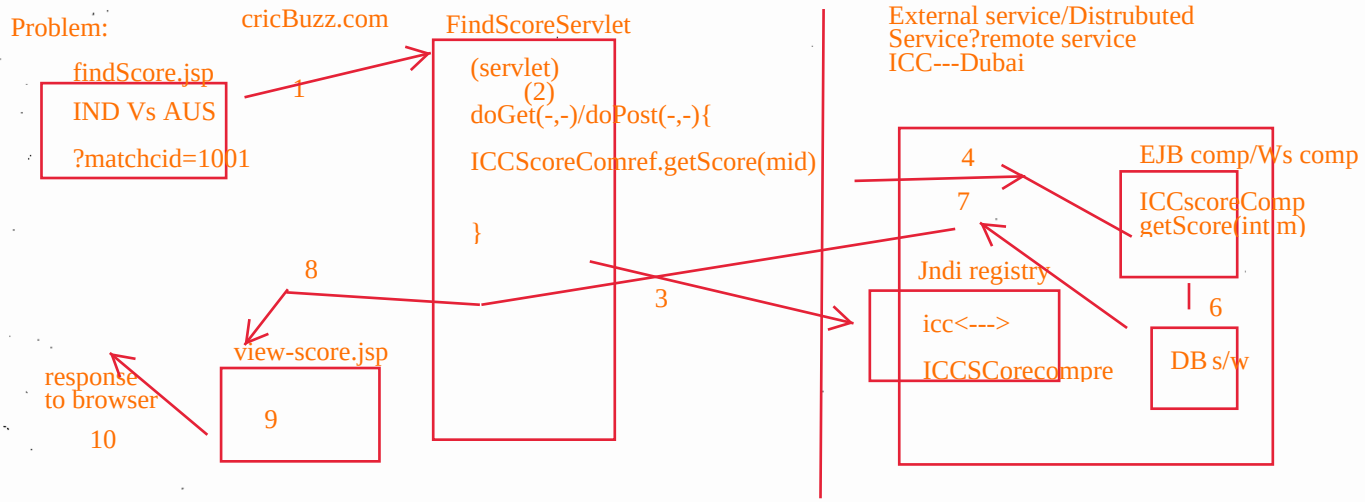
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
<display-name>IOCProj32Web-FactoryBeanSeviceLocatorUseCase</display-name>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>controller</servlet-name>
<servlet-class>com.nt.controller.FindScoreControllerServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>controller</servlet-name>
<url-pattern>/controller</url-pattern>
</servlet-mapping>
</web-app>
```

Index.jsp

```
<a href="controller?matchId=345"> IND Vs WI </a><br>
<a href="controller?matchId=567"> AUS Vs PAK </a><br>
```

View_score.jsp

```
<%@ page language="java" isELIgnored="false" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<h1><center> Score Card</center></h1>
Total Score ::::   ${scoreCard }
<br>
<a href="index.jsp">home</a>
```



Method Replacement

In normal classes if you want to replace existing logic of business method with new logics, we have 2 options.

a)Comment the existing logic and write the new logicin the same business method.

b)Develop the sub class and override business method with new logics.

In the above model we need to modify source code to revert back to original logics if we write new logics on experimental basis or if new logics exicution is failed in production environment at client side.The another problem with above approach is reverting back to original business logic is complex when source code is not exposed (.java code).

In retail industry offers will be placed for certain period, once the offer is over standard prices will be charged. To execute new logics during offer period and to revert back to original logics once the offer is over without touching the source code we can't use the above models.

To overcome the above problems use "method replacer/replacement" concept of spring. For this we need to develop separate class implementing "MethodReplacer(I)" and we need to write new logics in "*reimplement(-,-,-)*" method.

Public object reimplement (object object, Method method, Object [] args) *Object object*---->gives the access to original bean class object. *Method method*---->gives the access to original method that we want to replace. *Object [] args*---->gives access original method args.

We need to configure "MethodReplacer" class as spring bean and we need the IOC container to execute "*reimplement(-,-,-)*" when original method is called, for this we need to use *<replaced-method name="....." (Original Method) replacer="....." (bean id method replacer class)/>*

Method replacement is also called as method injection.

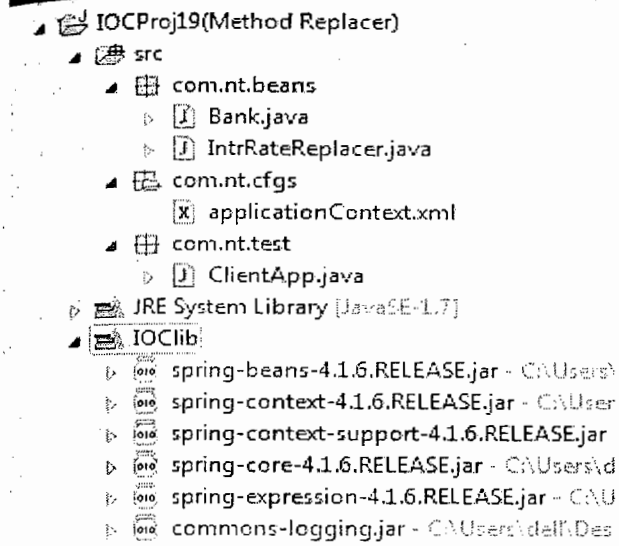
Note:

Every project goes through 2 phases:

1. Development and Testing phase.

2. Production phase (after release).

Example:



Bank.java

```
package com.nt.beans;
public class Bank {
    public  float calcIntrAmt(float pamt,float time){
            System.out.println("Bank:calcIntrAmt(-,-)");
            return pamt*time*2.0f/100;
    }
    public float calcIntrAmt(){
            return 123.0f;
    }
}
```

IntrRateReplacer.java

```
package com.nt.beans;
import java.lang.reflect.Method;
import java.util.Arrays;
import org.springframework.beans.factory.support.MethodReplacer;
public class IntrRateReplacer implements MethodReplacer {
    private float rate;

    public IntrRateReplacer() {
            System.out.println("IntrReteReplacer:0-param constructor");
    }
    public void setRate(float rate) {
            this.rate = rate;
    }
    @Override
    public Object reimplement(Object bean, Method method, Object[] args)
                    throws Throwable {
            System.out.println("IntrRateReplacer:reimplement(-,-,-)");
            System.out.println("bean class"+bean.getClass());
            System.out.println("Method name"+method.getName());
            System.out.println("Method args"+Arrays.toString(args));
            if(method.getName().equals("calcIntrAmt")){
                    float pamt=(Float)args[0];
```

```
                float time =(Float)args[1];
                return pamt*time*rate/100.0f;
        }
        else{
                return  0.0f;
        }
    }//reimplement
}//class
```

applicationContext.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.xsd">
<bean id="bank"  class="com.nt.beans.Bank">
<replaced-method  name="calcIntrAmt"  replacer="mr">
<arg-type>float</arg-type>
<arg-type>float</arg-type>
</replaced-method>
</bean>

<bean id="mr" class="com.nt.beans.IntrRateReplacer">
<property name="rate" value="1.2f"/>
</bean>
</beans>
```

When ctx.getBean(-,-) is called having beanId the container checks whether <replaced-method> is configured or not in the inmemory meta data of xml file. If configured container generates

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import com.nt.beans.Bank;

public class ClientApp {
 public static void main(String[] args) {
     //create IOC Container
            ApplicationContext ctx=new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext|
xml");
|
            //get Bean
|
            Bank bank=ctx.getBean("bank",Bank.class); ----------------------------------
--|
            System.out.println(bank.getClass());
            System.out.println("Intr Amt"+bank.calcIntrAmt(10000,20));  } }
```

Internals:

IOC container creates sub class for main bean class (*Bank*) like "*Bank$$CGLiB.....*" By using CGLib library internally, In this class business method (*calcIntrAmt()*) will be overridden. In this

method definition the object of method replacer class (*IntrRateReplacer*) will be accessed and reimplement method will be called.Due to this new logic will be executed when client application callsctx.*getBean ()* or *factory.getBean ()* the bellow generated subclass object will be returned.

Sample Code:

```
public class Bank$CGLiB extends Bank implements AppliactionContextAware
{
private ApplicationContext ctx ;
public void setApplicationContext (ApplicationContext ctx )
{                                   //crete IOC Container
                                    ctx=new ClassPathXmlApplicationContext("com/nt/cfgs");
this.cxt =ctx;
}
public float calcIntrAmt (-,-,-)
{
intrRateReplacer ir=ctx.getBean ("mr", IntrRateReplcer.class);
float retval=ir.reimplement (-,-,-);
return retval;
}
}
```

Important points :

1)The method that you want to replace should not be**"final, static"** (these methods can't be overridden in sublcass)

2)Eventhough you can write replacement logics for multiple method in single "reimplement()" method by using if conditions is recommended to take separate method replacer for every class.

3) We can't take bean class as final class while working with method replacer concepts.

Explain various injections supported by spring?

a)Setter injection

b)Constructor injection

c)Aware injection/context dependency lookup/ Interface injection

d)Method injection (method replacer)

e)Lookup method injection

```
**refer example lookup method internal 113:
MyServletContainer.java
public abstract class MyServletContainer{
   public abstract RequestHandler getRequestHandler();
     public void processRequest(String data){        (b)
      RequestHandler handler=null;
//get dependent class object
(f)     handler=getRequestHandler();  (c)
sysout("Request came to Container for processing----");
handler.handle(data);  (g)
}//method
}//class
Sub class of target class(inmemory class):
public class MyservletContainer$Proxy extends MyServletContainer implements ApplicationContextAware{
private ApplicationContext ctx;
public void setApplicationContext(ApplicationContext ctx){
this.ctx=ctx;
}
public RequestHandler getRequestHandler(){          (d)
  RequestHandler handler=null;
//dependency lookup
handler=ctx.getbean("handler",RequestHandler.class);
return handler;  (e)
}
}
client app:    MyservletContainer container=ctx.getbean("container";myservletcontainer.class);
container.processrequest("hello1");
```

Lookup method injection

- IOCProj20(LMInjection-problem)
  - src
    - com.nt.beans
      - MyServletContainer.java
      - RequestHandler.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - IOClib
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\c
    - spring-context-4.1.6.RELEASE.jar - C:\Users
    - spring-context-support-4.1.6.RELEASE.jar -
    - spring-core-4.1.6.RELEASE.jar - C:\Users\de
    - spring-expression-4.1.6.RELEASE.jar - C:\Us
    - commons-logging.jar - C:\Users\dell\Desk

MyServletContainer.java

```
package com.nt.beans;
public class MyServletContainer {
    private RequestHandler handler;
    public MyServletContainer() {
      System.out.println("MyServletContainer:0-param constructor");
    }
    public void setHandler(RequestHandler handler) {
            this.handler = handler;
    }
            public void processRequest(String data){
            handler.handle(data);
    }
}
```

RequestHandler.java

```
package com.nt.beans;
public class RequestHandler {
    private  static int count;
    public RequestHandler() {
            count++;
            System.out.println("RequestHandler: 0-param constructor");
    }
    public  void handle(String data){
            System.out.println("Handling Request "+count+" with data "+data);
    }}
```

Note: we can configure abstract class to spring bean
a: when lookup method injection in enable to bean class
b: when factorymethod bean instantiation is enable on bean class
Note: when abstract class configure as spring bean container does not create object for abstract class. but it creates objects for dynamically generator inmemory class or object for class that is given by factory method

applicationContext.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.xsd">
<bean id="rh" class="com.nt.beans.RequestHandler" scope="prototype"/>
<bean id="container" class="com.nt.beans.MyServletContainer" scope="singleton">
<property name="handler" ref="rh"/>
</bean>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.MyServletContainer;
public class ClientApp {
    public static void main(String[] args) {
            // IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //get Bean
            MyServletContainer
container=ctx.getBean("container",MyServletContainer.class);
            container.processRequest("hello");
            container.processRequest("hai");
    }
}
```

Problem with dependency injection:

When prototype scope bean is dependent to singleton scope bean somehow the prototype scope bean also acts as singleton scope bean.

eg: All request coming to webApplication will be handled by single servlet container/ web container (Singleton).

This container uses separate "RequestHanlder" to process each request so servlet container is target class and "RequestHandler" is dependent class, moreover ServletContainer should be taken as singleton scopebean and "RequestHandler" should taken as prototype scope, but because of above problem "RequestHandler" also acts as the singleton scope bean.

```
<bean id="container" class="com.nt.beansMyservletcontainer" scope="singleton">
<property name="handler" ref="rh"/>
</bean>
<bean id="rh" class="com.nt.beans.RequestHandler" scope="prototype"/>
```

Target Bean Scope	Dependent Bean Scope	Resultant Scope of Dependent Bean
Prototype	Singleton	Singleton (OK)
Singleton	Singleton	Singleton (OK)
Prototype	Prototype	Prototype (OK)
Singleton	Prototype	Singleton (Not OK) (use dependency lookup or-Lookup method injection)

IOCProj21(LMInjection-sol1)

RequestHandler.java

```
package com.nt.beans;
publicclass RequestHandler {
	privatestaticintcount;
	public RequestHandler() {
		count++;
		System.out.println("RequestHandler: 0-param constructor");
	}
	publicvoid handle(String data){
		System.out.println("Handling Request"+count+"with data"+data);
	}
}
```

MyServletContainer.java

```
package com.nt.beans;
import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
publicclass MyServletContainer implements ApplicationContextAware {
	ApplicationContext ctx;
	private String beanId;
	publicvoid setBeanId(String beanId) {
		this.beanId = beanId;
	}
	public MyServletContainer() {
		System.out.println("MyservletContainer:0-param constructor");
	}
	publicvoid processRequest(String data){
		//get Dependent Bean
		RequestHandler handler = ctx.getBean(beanId,RequestHandler.class);
		handler.handle(data);
	}
	publicvoid setApplicationContext(ApplicationContext ctx)
			throws BeansException {
		this.ctx = ctx;
	}
}
```

applicationContext.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.xsd">
    <bean id="rh1" class="com.nt.beans.RequestHandler" scope="prototype"/>
    <bean id="container" class="com.nt.beans.MyServletcontainer" scope="singleton">
        <property name="beanId">
                <idref bean="rh1"/>
    </property>
    </bean>
    </beans>
```

ClientApp.java

```
packagetest;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beansMyServletContainer;
publicclass ClientApp {
        publicstaticvoid main(String[] args) {
                //IOC Container
                ApplicationContext ctx = new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationcontext.xml");
                //get Bean
MyServletContainer container = ctx.getBean("container",MyServletContainer.class");
container.processRequest("hello");
container.processRequest("hai");
        }
}
```

IOCProj22(Lookup method Injection – Final Solution)

RequestHandler.java

```
package com.nt.beans;
publicclass RequestHandler {
        privatestaticintcount;
        public RequestHandler() {
                count++;
                System.out.println("RequestHandler: 0-param constructor");
        }
        publicvoid handle(String data){
                System.out.println("Handling Request"+count+"with data"+data);
        }
}
```

MyServletContainer.java

```
package com.nt.beans;
publicabstractclass MyServletContainer {
        public MyServletContainer() {
                System.out.println("MyServletContainer:0-param constructor");
        }
        publicvoid processrequest(String data){
                //get Dependent Bean
                RequestHandler handler = getRequestHandler();
                handler.handle(data);
        }
        publicabstract RequestHandler getRequestHandler();
}
```

applicationContext.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.xsd">
    <bean id="rh" class="com.nt.beans.RequestHandler" scope="prototype"/>
    <bean id="container" class="com.nt.beans.MyServletcontainer" scope="singleton">
        <lookup-method name="getRequestHandler" ref="rh"/>
    </bean>
    </beans>
```

by seeing this tag ioc container generator inmemory sub class for target class and create object and assign for target class bean id to that object.

ClientApp.java:

```
package com.nt.beans;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
publicclass ClientApp {
        publicstaticvoid main(String[] args) {
                //IOC Container
                ApplicationContext ctx = new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationcontext.xml");
                //get Bean
                MyServletContainer
container=ctx.getBean("container",MyServletContainer.class);
                container.processrequest("hello");
                container.processrequest("hai");
                System.out.println(container.getClass());
        }
}
```

We should not think about dependency injection (constructor/setter) in the following situations.

a)If dependent bean class object is required in target Bean only in specific methods

b)If target Bean scope is singleton and Dependent Bean scope is prototype (refer above [illegible]

In the above situation prefer using Dependency lookup

a)If we go for traditional dependency lookup code we need to create extra IOC container in the ~~above~~ specific methods of target class (which is bad practice) **

b)By using aware injection we can avoid that extra IOC container towards dependency lookup operational. But it kills the non-invasive behavior of spring Application because target bean should implement "xxxAware"interfaces.

c)To overcome the above two problems we can use "Lookup method Injection (LMI)" which says performs "dependency lookup" but container should write code for it (not the programmer).

** when client application already having IOC container taking one more ioc container in target class method is really bad practice Due to this singlton scope target bean looks like prototype scope bean.

To perform Lookup Method Injection we need to consider the following points:

a)Target bean class should have abstract method returning Dependent Bean class (for this target bean class must be taken as abstract class)

b)We must configure the above abstract method with targetBean class configuring by using *<lookup-method>* tag

Note:

We can configure abstract class as spring bean only when *<lookup-method>* tag is placed for lookup method injection otherwise not possible. But while working with facrory method bean instantiation we can also take abstract class as spring bean.

Spring IOC container generates subclass for targetBean class and implements the abstract method of that class to get and Return dependent bean class object. For this it internally uses CGLIB Library.

In spring 2.x,3.x we need to supply CGLIB Library separately where as in spring 4.x, not required.

Lookup method injection use cases

ServletContainer (Singleton)<---------->Request Handler (prototype)

Manager(Singleton)<---------->Customer (prototype)

LoanApprover (Singleton)<---------->LoanDetails (prototype).

Internal code for the above final solution example: refer 107 page:

BeanPostProcessor (Bpp)

Assigning some values to bean properties after injection is called Post processing. We can do this post processing inside the bean class and specific to bean class by using custom*init()* method or (InitializingBean's) *afterPropertiesSet()* of bean class.If we want to apply common post processing logic for multiple beans that to being from outside the bean class then we can use BeanPostProssor class.

The "BeanPostProcessor" class executes for all beans that are configured in spring bean configuration file.To develop BeanPostProcessor we need to take a class that implements "BeanPostProcessor(I)" and should provide implementation for 2 methods.

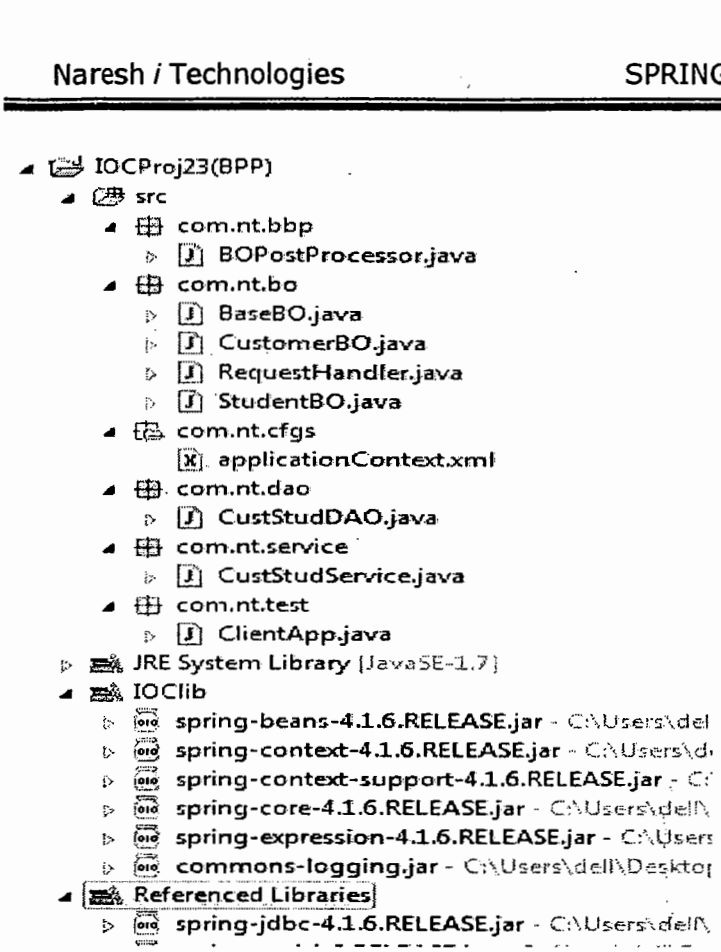
a)PostProcessorBeforeInitialization(Object bean, Spring Beanid)

Executes before *afterPropertiesSet()* for every beans class.

c)PostProssorAfterInitialization (Object bean, Spring bean id)

Executes after custom *init()* for every bean class.

The concept is useful to develop custom post process/initialization logic for all beans in one class.



BaseBO.java

```
package com.nt.bo;
import java.util.Date;
public abstract class BaseBO {
    private int id;
    private String name;
    private  Date doj;
    public BaseBO() {
      System.out.println("BaseBO:0-param constructor");
    }
    public void setDoj(Date doj) {
            this.doj = doj;
    }
    public void setId(int id) {
            this.id = id;
    }
    public void setName(String name) {
            this.name = name;
    }
    public int getId() {
            return id;
    }
    public String getName() {
            return name;
    }
    public Date getDoj() {
            return doj;     }}
```

CustomerBO.java

```
package com.nt.bo;
public class CustomerBO extends BaseBO {
    private float billAmt;
    public CustomerBO() {
      System.out.println("CustomerBO:0-param cosntructor");
    }
    public float getBillAmt() {
            return billAmt;
    }
    public void setBillAmt(float billAmt) {
            this.billAmt = billAmt;
    }
}
```

StudentBO.java

```
package com.nt.bo;
public class StudentBO extends BaseBO {
    private String course;
    public StudentBO() {
      System.out.println("StudentBO:0-param constructor");
    }
    public String getCourse() {
            return course;
    }
    public void setCourse(String course) {
            this.course = course;
    }
}
```

BOPostProcessor.java

```
package com.nt.bpp;
import java.util.Date;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import com.nt.bo.BaseBO;
public class BOPostProcessor implements BeanPostProcessor {
    public BOPostProcessor() {
      System.out.println("BPP: 0-param constructor");
    }
    @Override
    public Object postProcessBeforeInitialization(Object bean, String bid)
                    throws BeansException {
      System.out.println("BPP:BeforInitialization(-,-)");
      if(bean instanceof BaseBO){
    ((BaseBO) bean).setDoj(new Date());
    return bean;
      }
            return bean;
    }
    @Override
    public Object postProcessAfterInitialization(Object bean, String bid)
                    throws BeansException {
            System.out.println("BPP:AfterInitialization(-,-)");
            return bean;
    }
}
```

CustStudDAO.java

```
package com.nt.dao;
import com.nt.bo.CustomerBO;
import com.nt.bo.StudentBO;
public class CustStudDAO {
   public CustStudDAO() {
    System.out.println("DAO:0-param constructor");
}
    public void  insertData(StudentBO stbo){
            System.out.println("Student Data is");
            System.out.println(stbo.getId()+" "+stbo.getName()+" "+stbo.getCourse()+"
    "+stbo.getDoj());
    }
    public void  insertData(CustomerBO custbo){
            System.out.println("Customer Data is");
            System.out.println(custbo.getId()+" "+custbo.getName()+"
"+custbo.getBillAmt()+"      "+custbo.getDoj());
    }
}
```

CustStudService.java

```
package com.nt.service;
import com.nt.bo.CustomerBO;
import com.nt.bo.StudentBO;
import com.nt.dao.CustStudDAO;
public abstract class CustStudService {
    private CustStudDAO dao;

    public CustStudService() {
     System.out.println("Service : 0-param constructor");
    }
    public abstract StudentBO  getStudentBO();
     public abstract CustomerBO  getCustomerBO();
     public void setDao(CustStudDAO dao) {
                this.dao = dao;
            }
     public void processStudent(int id,String name,String course){
            System.out.println("Procesing Student Info");
            StudentBO  stbo=getStudentBO();
            stbo.setId(id); stbo.setName(name); stbo.setCourse(course);
            //use dao
            dao.insertData(stbo);
     }
     public void processCustomer(int id,String name,float billAmt){
            System.out.println("Procesing Customer Info");
            CustomerBO  custbo=getCustomerBO();
            custbo.setId(id); custbo.setName(name); custbo.setBillAmt(billAmt);
            //use dao
            dao.insertData(custbo);
                }
}
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.CustStudService;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");

    /*      BeanFactory factory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
            //register BeanPostProcessor
            BOPostProcessor bpp=new BOPostProcessor();
            ((ConfigurableListableBeanFactory)factory).addBeanPostProcessor(bpp); */
```

```
            //get Bean
    CustStudService service=ctx.getBean("service",CustStudService.class);
            //call b.methods
            service.processCustomer(1001,"raja",9000);
            service.processStudent(101,"karan","java");
            service.processCustomer(1002,"raja",9000);
            service.processStudent(102,"karan","java");
    }
}
```

applicationContext.xml

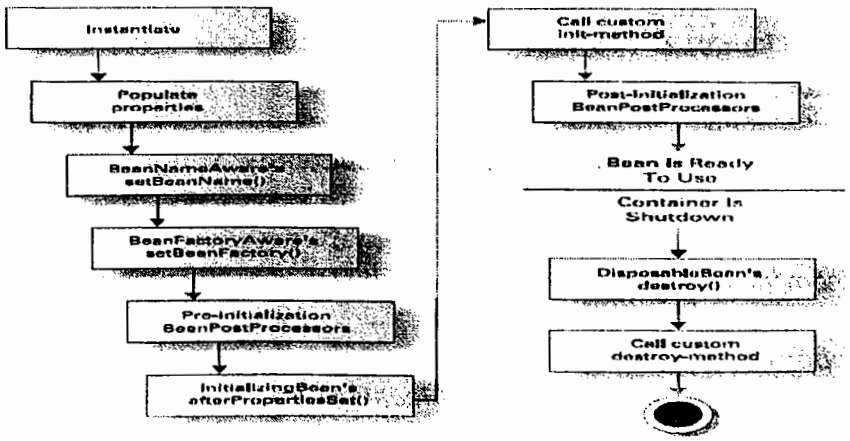
```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="stBO" class="com.nt.bo.StudentBO" scope="prototype"/>
<bean id="custBO" class="com.nt.bo.CustomerBO" scope="prototype"/>
<bean id="dao"  class="com.nt.dao.CustStudDAO" scope="singleton"/>

<bean id="service" class="com.nt.service.CustStudService" scope="singleton">
<property name="dao" ref="dao"/>
<lookup-method name="getStudentBO"  bean="stBO"/>
<lookup-method name="getCustomerBO"  bean="custBO"/>
</bean>
<bean id="bpp"  class="com.nt.bpp.BOPostProcessor"/>
</beans>
```

In real time *BeanPostProcessor* will be used to place common post process initialization logic that is required for multiple Beans like assigining system date as date of joining or last modification date, date of updation and etc.....

This concept is very useful towards auditing activity.While working with *ApplicationContext* container if we just configure *BeanPostProcessor* class in Xml file as spring Bean the container automatically recognize and use that *BeanPostProcessor*.While working with BeanFactory container we must have to register BeanPostProcessor explicitly as shown below

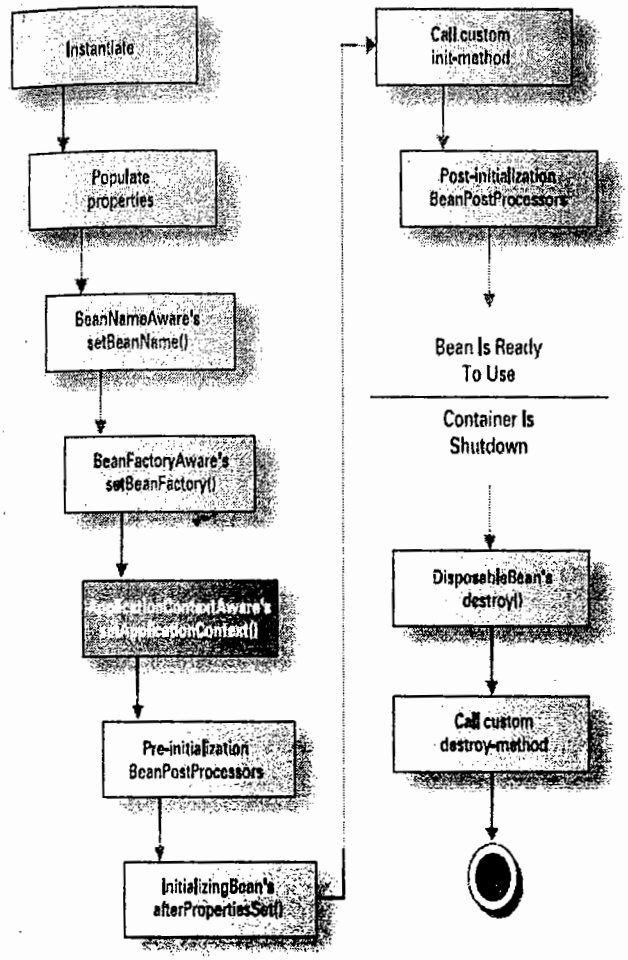
BOPostProcessor bpp=new BOPostProcessor (); //Bpp Lass
((ConfigurableListableBeanFactory)Factory).addBeanPostProcessorc(bpp);

Bean Life Cycle diagram with respect to BeanFactoryContainer:



If the underlying container is ApplicationContext container it can inject both BeanFactory, ApplicationContext objects to spring bean through Aware injection.

Bean Life Cycle diagram with respect to ApplicationContext container:



Complete Internal flow of core module application:

a)Programmers creates *BeanFactory* object.

b)Given SpringBean configuration file will be located and it will be verified whether document is well formed document or valid document .

c)Creates Inmemory logical IOC container in the jvm's memory.

d)Reads springBean configuration file entries and creates an inmemory metadata having all the configuration's.

e)ClientApp calls *factory.getBean("id")* method.

f)Loads Bean class from inmemory meta data based on given bean id and instantiates that bean based on constructor injection configuration.

g)Keep Bean class objects in specified scope (if cyclic dependency is found Bean Instantiation fails)

h)Performs setter injection based on configuration by performing dependency-check.

i)Performs Aware injection based on the *xxxAware* interfaces that are implement.

j)Calls *postProcessBeforeInitialization(-)* if *BeanPostProccessor* is added.

k)Calls *afterPropertySet(-)* if *initializationBean(i)* implemented.

l)Calls custom init method if *"init-method"* attribute is specified.

m)Calls*postProcessAfterInitialization()*

n)Returns Bean class object to client application.

Note: If ApplicationContext object is create (f) to (n) also takes place without calling getBean(-) for singleton scope beans.

Note: if <lookup-method> or <replaced-method> tags are configured inside <bean> tag then container does not create object given bean class. It creates object for the sub class of the bean class. Which is generated by IOC container.

Note: If Bean class FactoryBean (I) or factory-method attribute is configured in the <bean> tag then container doesn't create given bean class object. Container creates /gets resultant object and keeps that resultant object with the configured beanId.

BeanFactoryPostProcessors

BeanPostProcessor is useful to write postProcess initialization logics with respect to multiple beans.

BeanFactoryPostProccessor is useful to add extra configurations by modifying inmemory metadata of bean configured file after the creation of IOC container and before accessing bean object by calling *getBean(-)*

It acts as extension hook/anchor for IOC container to add extra configurations without touching the xml file. (Springbean configuration file) i.e it modifies the inmemory metadata of IOC container dynamically at runtime after IOC container creation and before bean class object is accessed.

To develop custom BeanFactoryProcessor we need to take a class implementing "*BeanFactoryPostProcessor(I)*". Spring has given lots of pre-defined"*BeanFactoryPostProcessor(I)*" like '*PropertyPlaceholderConfigurer*'. This class modifiers in memory metadata by replacing placeholder *${--}* with the values collected from properties File. "*ApplicationContext*" object based IOC container automatically recognizes the BeanFactoryPostProcessor that is configured in xml file But for BeanFactory object based IOC container we need to register explicitly.

By default the IOC container of "*BeanFactory*" can't work with properties file and place holder. But we can make it working by registering "*PropertyPlaceholderConfigurer*" as *BeanFactoryPostprocessor*

Example:

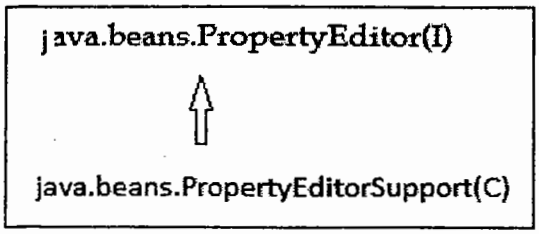
```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="pcc" class="org.gpr.prk.bean.Factory.config.PlaceholderConfigure">
<Property name="location" value="com/nt/commons/DBdetails.properties"/>
</bean>
</beans>
```

ClientApp:

```
BeanFactory factory=new XmlBeanFactory (new FilesystemResource
("src/com/nt/cfgs/applicationContext.xml"));
//get propertyplaceholder configure object (pphc)
BeanFactoryPostProcessor bfpp=factory.getBean ("PPC" PropertyPlaceholderConfigurer.class);
//register
bfpp.postProcessBeanFactory((ConfigurableListableBeanFactory)Factory);
```

PropertyEditor

In spring we use setter or constructor injection to assign values to bean properties. In spring Bean configuration file we configure these dependent values as String values, but they will converted into *int,long,float,Date* and etc. values as required for bean properties automatically. For this spring internally uses *PropertyEditors* to convert string values to appropriate values.All property editors are the classes implementing *java.beans.PropertyEditor(I)*.Spring has given multiplebuilt-in *PropertyEditors* like StringArraryPropertyEditors, *ByteArraryPropertyEditor,DateFormatePropertyEditor* and etc....(FileEditor,CharacterEditor)

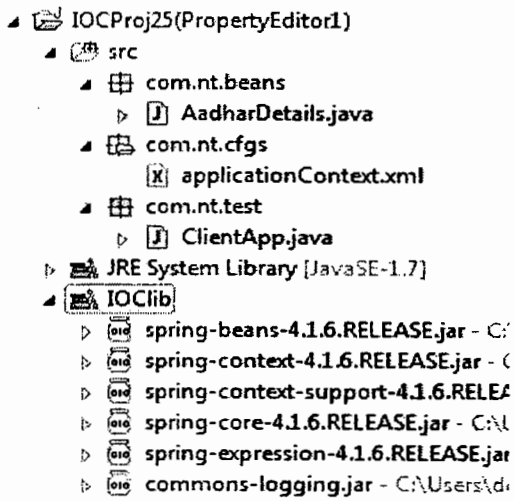


PropertyEdito hierarchy

Note:

Registrar maintains registry and we can register one (or) more property editor objects with registry.

Understanding Built-in property editors.



AadharDetails.java

```
package com.nt.beans;
import java.io.File;
import java.util.Arrays;
import java.util.Date;
public class AadharDetails {
        private int  uid;
       private String name;
       private Date dob;
       private File   photo;
       private String[] verifiers;
      public void setUid(int uid) {
              this.uid = uid;
      }
      public void setName(String name) {
              this.name = name;
      }
      public void setDob(Date dob) {
              this.dob = dob;
      }
      public void setPhoto(File photo) {
              this.photo = photo;
      }
```

```
    public void setVerifiers(String[] verifiers) {
        this.verifiers = verifiers;
    }
    @Override
    public String toString() {
        return "AadharDetails [uid=" + uid + ", name=" + name + ", dob=" + dob
                + ", photo=" + photo + ", verifiers="
                + Arrays.toString(verifiers) + "]";
    }
}
```

applicationContext.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.xsd">
<bean id="aadhar" class="com.nt.beans.AadharDetails">
<property name-="uid" value="101"/>
<property name-="name" value="Raja"/>
<property name-="dob" value="12/11/1990"/>
<property name-="Photo" value="C:/abd/picd.gif"/>
<property name-="verifiers" value="sahin,Ramesh,Ravi"/>
</bean>
</beans>
```

Note: In both containers(BF,AC) Built-in property editors are automatically registered. Spring Internally uses "DateFormatePropertyEditor" to convert given date value to *java.util.Date* class object.Spring allows to create custom property editor class either by implementing *java.beans.PropertyEditor(I)* or by extending from PropertyEditorSupport(c).

We need to use CustomEditorConfigurer class to register custom property editor while working with ApplicationContextContainer.

In applicationContext.xml

```
<bean class="org.springframework.beans.factory.config.CustomEditorConfigurarer">
<property name="customEditors">
<map>
<entry key="com.nt.beans.IntrAmtDetails"
            value="com.nt.editor.IntrAmtDetailsEditor"/>
</map>
</property>
</bean>
```

IntrAmtDetails.java

```
public class IntrAmtDetails
{
private float p,t,r;
//setters
}
IntrAmtCalculator.java
public class IntrAmtCalculator
{
privator  IntrAmtDetails details;
//setters
.................}
```

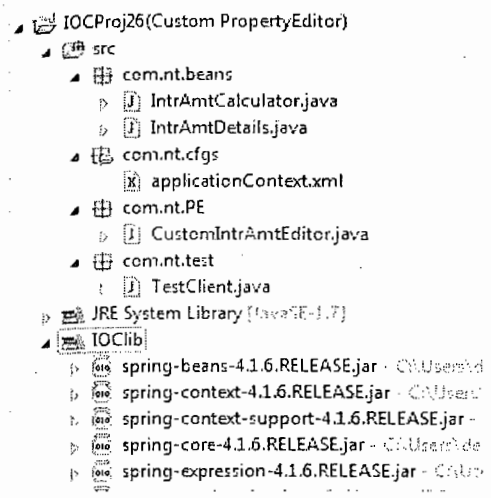
InapplicationContext.xml

```
<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.xsd">
<bean id="amtdetails" class="pkg.IntrAmtDetails"/>
<property name="P" value="1000"/>
<property name="t" value="10"/>
<property name="r" value="2"/>
</bean>
<bean id="intrCalculator" Class="pkg.IntrAmtCalculator">
<property name="details" ref="amtdetails"/>
</bean>
</beans>
```

But we want like this
applicationContext.xml:

```
<bean id="intrCalculator" class="pkg.IntrAmtCalculator">
<property name="details" values="1000,10,2"/>
</bean>
```

Here "details" to reference type property but we want to perform simple values based injection which is not possible directly. So must go for custom property editor development.




```
package com.nt.beans;
```

IntrAmtCalculator.java

```
public class IntrAmtCalculator {
        private IntrAmtDetails details;
        public void setDetails(IntrAmtDetails details) {
        this.details = details;
    }
        public float calcIntrAmt(){
        return (details.getPrinciple()*details.getRate()*details.getTime())/100.0f;
    }
}
```

IntrAmtDetails.java

```
package com.nt.beans;
public class IntrAmtDetails {
    private int principle;
    private int time;
    private int rate;
    public void setPrinciple(int principle) {
        this.principle = principle;
    }
    public void setTime(int time) {
        this.time = time;
    }
    public void setRate(int rate) {
        this.rate = rate;
    }
    public int getPrinciple() {
        return principle;
    }
    public int getTime() {
        return time;
    }
    public int getRate() {
        return rate;
    } }
```

applicationContext.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.xsd">

    <!--  <bean id="amtDetails"  class="com.nt.beans.IntrAmtDetails">
    <property name="principle" value="1000"/>
    <property name="time" value="10"/>
    <property name="rate" value="2"/>
    </bean>
    <bean id="intrCalculator"  class="com.nt.beans.IntrAmtCalculator">
    <property name="details" ref="amtDetails"/>
    </bean> -->
    <bean id="intrCalculator"  class="com.nt.beans.IntrAmtCalculator" scope="prototype">
    <property name="details"  value="1000,10,2"/>
    </bean>
    </beans>
```

CustomIntrAmtEditor.java

```
package com.nt.PE;
import java.beans.PropertyEditorSupport;
import com.nt.beans.IntrAmtDetails;
public class CustomIntrAmtEditor extends PropertyEditorSupport {
    public CustomIntrAmtEditor() {
      System.out.println("CustomIntrAmtEditor:0-param constructor");
    }
    @Override
    public void setAsText(String text) throws IllegalArgumentException {
            System.out.println("CustomIntrAmtEditor:setAsText(-)");
            // spilt the given value
            int principle=Integer.parseInt(text.substring(0,text.indexOf(",")));
            int
time=Integer.parseInt(text.substring(text.indexOf(",")+1,text.lastIndexOf(",")));
            int rate=Integer.parseInt(text.substring(text.lastIndexOf(",")+1,text.length()));
        IntrAmtDetails details=new IntrAmtDetails();
        details.setPrinciple(principle); details.setTime(time); details.setRate(rate);
        setValue(details);
    }//method
}//class
```

TestClient.java

```
package test;
import org.springframework.beans.PropertyEditorRegistrar;
import org.springframework.beans.PropertyEditorRegistry;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
```

```
import com.nt.PE.CustomIntrAmtEditor;
import com.nt.beans.IntrAmtCalculator;
import com.nt.beans.IntrAmtDetails;
public class TestClient {
    public static void main(String[] args) {
            //Create IOC container
            BeanFactory factory=new XmlBeanFactory(new
    FileSystemResource("src/com/nt/cfgs/applicationContext.xml"));
  ((ConfigurableListableBeanFactory)factory).addPropertyEditorRegistrar(new
MyCustomPE());
            //get Bean
            IntrAmtCalculator
iCalc=factory.getBean("intrCalculator",IntrAmtCalculator.class);
            System.out.println("Intr Amt"+iCalc.calcIntrAmt());
            }//main
}//class
 class MyCustomPE implements PropertyEditorRegistrar{
    @Override
    public void registerCustomEditors(PropertyEditorRegistry registry) {
            registry.registerCustomEditor(IntrAmtDetails.class,new CustomIntrAmtEditor());
            }
    }
```

Every IOC container internally maintains one builtin object called "PropertyEditorRegistry" and IOC container expose that object to a class that implements "PropertyEditorRegistrar" interface as the arugment of '*registerCustomEditors()*' method so that we can use that object to register CustomPropertyEditor by specifying property type and the object of CustomPropertyEditor.PropertyEditorRegistrar is an object maintained by IOC container and IOC container exposes the internally maintained PropertyEditorRegistry object to its implementation class.

we can create user define propertyeditor registrar by taking class and implementing propertyeditorregistrar interface in case of applicationcontext container in above registration process is not required because we can use one predefine beanfactory postprocessor custom editor configurer for the sake as ulternate
refer page no.123(applicationContext.xml

Spring Core Module with Annotations:

Annotations: Data about data and code about code is called Metadata. [illegible] are java statements to perform instructions to [illegible] files. Annotations [illegible] for xml files to perform metadata, resource configuration. And the [illegible] of modification but [illegible] xml documents.

Annotations do not provide flexibility of modification but provide good performance.

Syntax: @<Annotation name> (param1=val1 ...)

Annotations for documentation are introduced from jdk 1.0 and annotations for programming are introduced from jdk 1.5. We can apply annotations at different level like class level, Method level, field level, Param level and etc..The annotations will be used by compilers, [illegible], AnnotationProcessors, JRE and etc..

Some annotations provides instructions in the generation of api documents.

Eg:*@Author, @Since* and etc...

Some annotations will provide instruction to compiler but no effect will be there at runtime.

Eg:*@Override (Tells compiler it is inherited method of class)*

Some annotations are useful to mark the classes/interfaces as special items.

Eg:@Entity- makes ordinary JavaBean as Hibernate domain class. *@WebServlet -* makes class as servlet comp and etc...

Some annotations are useful to generate new classes using tools dynamically based on given class code.

Eg:@WebService

All the java technologies, Frameworks that are designed based on jdk1.5+ are supporting annotations.

Eg: Servlet 3.X+, hibernate 3.2X+, Spring2.X+, EJB3.0+, and etc..

If we perform certain settings by using annotations and xml file th[illegible]ngs done in xml file will override the settings done through annotations.

Spring annotations: (For core module)

Spring 2.0 onwards there is a support for annotation and more annotations are added incrementally in next version.In spring programming we need to use both xml and annotations together because we can instruct IOC container to recognize the used annotations only through xml file.We cannot use annotations to make predefined classes as the spring beans because we can't add annotations by opening the source code in those class.To configure such classes as spring beans we need to work with xml support.

Spring 2.0

@Configuration @Required @Repository

Spring 2.5

@Autowired @Qualifier @Scope

Stereotype annotations:

@Component @Service @Controller

Spring 3.0

@Bean *@Depends on* *@Lazy* *@Value*

Java config annotations (can be used only with spring3.X):

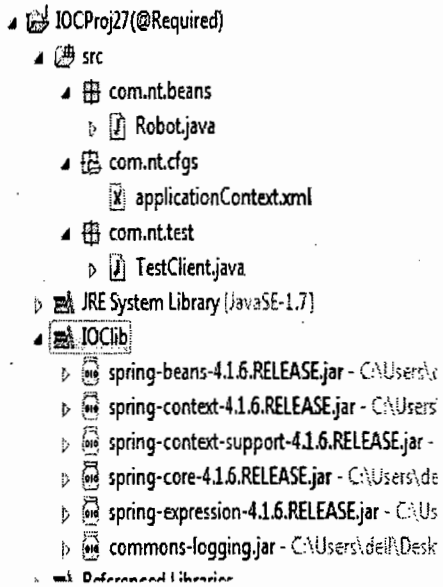
@PostConstruct *@PreDestruct* *@Resource* *@Inject* *@Named*

Spring 3.x:- @componentScan @imort @importResource @propertiesSource
@componentscan: to specify packages where annotated spring beans are available
@import: to import one bean configurator class to another bean configurator class
@importSource: to import xml spring configuration file to bean configurator class
@propertiesSource: to specify properties file

@Required:

To make dependency injection (setter injection) mandatory on properties we can use dependency check activity. But this is removed from spring2.5 because it does not allow the programmer to enable the restriction on specific properties. It allows the programmer to enable restriction on group of properties like "simple", objects, all". To overcome this problem we can use *@Required*

eg: Robot.java



Spring-aop-RELEASE.jar(also add this jar to IOClib)

Robot.java

```
package com.nt.beans;
import org.springframework.beans.factory.annotation.Required;
public class Robot{
    private int id;
    private String type;
    public void setId(int id){
        this.id=id;
    }
    @Required
   public void setType(String  type){
      this.type=type;
   }
    //toString()
@Override
public String toString() {
     return "Robot [id=" + id + ", type=" + type + "]";
}
     }
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean id="robot" class="com.nt.beans.Robot">
<property name="id" value="1001"/>
<!--  <property name="type" value="Army"/> -->
```

```
    </bean>
    <context:annotation-config/>
    // (or)
<bean
class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/
>
    </beans>
```

executes after bean initialization and provide functionality further @requiredAnnotation

<context:annotation-config>

The above tag makes IOC container to recognize the annotation and to provide functionality for the *@Required* annotation.

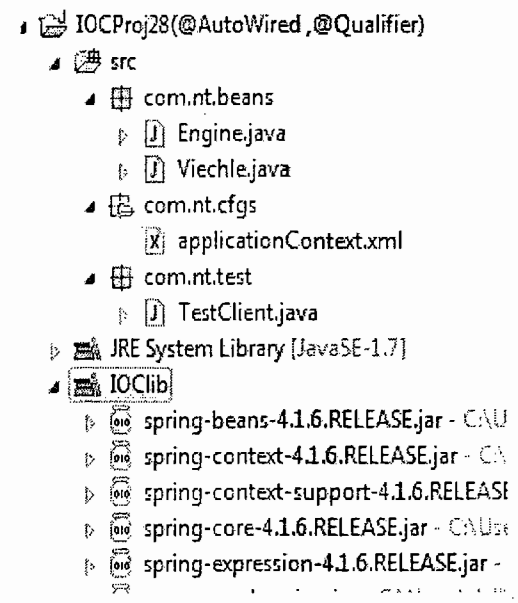
ClientApp.java:

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Robot;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.
xml");
            //getBean
            Robot robo=ctx.getBean("robot",Robot.class);
            System.out.println(robo);
    }
}
```

The "BeanPostPorcessor" of every annotation contains logic to recognize the annotation and to provide functionality for the annotation. But configuring the BeanPostProcessor of every annotation is complex process. To overcome this problem use ***<context:annotation-config/>*** tag. To work with this tag import schema namespace in the xml.

@Autowired:

Given to perform byType, byName, constructor mode of autowiring. **This annotation can be applied at field level, setter method level, constructor level and arbitrary method level.**



Engine.java:

```
package com.nt.beans;
public class Engine{
        private int id;
        private String type;
        //setters
      public int getId() {
              return id;
      }
      public void setId(int id) {
              this.id = id;
      }
      public String getType() {
              return type;
      }
      public void setType(String type) {
              this.type = type;
      }
         //toString()
      @Override
      public String toString() {
              return "Engine [id=" + id + ", type=" + type + "]";
      }
              }
```

Vehicle.java

```
package com.nt.beans;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class Viechle{
    @Autowired
    @Qualifier("e")
     private Engine engg;
    /* @Autowired(required=true)
     public Viechle(Engine engg){
            System.out.println("1-param constructor");
            this.engg=engg;
     }*/
/*   @Autowired(required=true)
     public void setEngg(Engine engg){
            System.out.println("setEngg");
                    this.engg=engg;
     }
            @Autowired(required=true)
     public void assign(Engine engg){
            System.out.println("assign");
            this.engg=engg;
 }*/
     }*/
@Override
    public String toString() {
            return "Viechle [engg=" + engg + "]";
    }           }
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context.xsd">
```

```
   <bean id="eng"  class="com.nt.beans.Engine">
   <property name="id" value="11"/>
   <property name="type" value="CRDI"/>
   <qualifier value="e"/>
   </bean>
   <bean id="eng1"  class="com.nt.beans.Engine">
   <property name="id" value="12"/>
   <property name="type" value="DDIS"/>
   <qualifier value="e1"/>
   </bean>
   <bean id="viechle" class="com.nt.beans.Viechle"/>
   <context:annotation-config/>
```

```
</beans>
```

ClientApp:

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Viechle;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //getBean
            Viechle viechle=ctx.getBean("viechle",Viechle.class);
            System.out.println(viechle);
    }
}
```

While working with ***<context:annotation-cnfig>*** tag we need to add"*spring-aop<version>*".Release .jar file as an extra jar file to build path.

Vehicle.java (at settermethod level)

```
public class vehicle
{
    private Engine engg;
@Autowired (required=true)
public void setEngg (Engine engg)
{
this.engg=engg;
}
.................
.................
}
```

@Autowired (required=true)

If dependent bean is not available to perform dependency injection through autowiring exception will be raised when required is true. Default value is "true". Null will be assigned if *required="false"* and dependent bean configuration is not available. (i.e no exception will raise)

Autowiring by using arbitrary method:

```
public class vehicle
{
private Engine engg;
@Autowired (required=true)
public void assign (Engg engg)  //arbitrary method (your choice method)
{
    this.engg=engg;
}
    //toString()
.....
.....
}
```

Autowiring by using constructor:

```
public class Vehicle
{
private Engine engg;
@Autowired (required=true)
public vehicle (Engg engg)
{
    this.engg=engg;
}
    //toString()
.................
.......................
}
```

Note:
We can place @Autowired annotation on the top of only one parameterized constructor at a time. But the same annotation can be applied on multiple fields, setter methods, arbitrary methods at a time.

We cannot apply *@Autowired* on the top of O-param constructor

At max only 1 constructor can have *@Autowired* annotation.When *@Autowired* is placed on the top of field or setter method or arbitrary method. Then "*byType*" mode autowiring is takes place. When the same annotation is placed on top of constructor then constructor mode autowired will take place.

@Qualifier:
While performing "*byType*" mode autowiring there is a possibility of getting ambiguity if multiple dependents are found to inject to target class. Inorder to overcome this problem we need to use "*@Qualifier*" annotation which makes IOC container to perform "*byName*" mode of auto-wiring.

@Component, @Service, @Repository, @Controller:
The above annotations are called stereo type annotation, because all the 4 types annotations are having same purpose with minor changes.We can use all the 4 annotations to configure java classes as the spring beans.

@Component:
Generic purpose stereo type annotation to configure any java class as spring bean. i.e exposes given java class as spring bean to IOC container.

@Service:
Specialization of *@Component*, useful to configure the java classes of service layer. As of now no special behavior, so it is same as *@Component*

@Repository:
Specialization of *@Component*, useful to config the DAO classes as the Spring bean. Capable of translating one form of exception to another form of exceptions (useful to translate persistence technology exception to another exception).

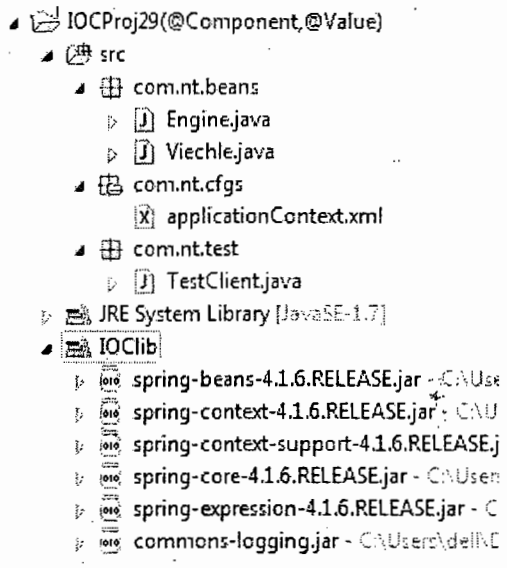
@Controller:
Specialization of *@Component*, use to configure classes as the controller of spring mvc app to process request given by clients.

Note:
1) If annotation parameter name is "value" we can set value/data to that parameter without specifying the parameter name (value).

2)<context:component-scan base-package="----------"/> searches in the given packages to take the classes that are annotated with stereo type annotations as spring beans and also uses

some BeanfactoryPostProcessor to update the inmemory meta data by having these loaded classes.It internally contains the behavior of <context:annotation-config/>. So this tag is not required while working with<contex:component-scan> tag.

eg: while working with annotation if you don't provide bean id then it takes bean class name as default bean id
ex:- bean class name is Flipkart then it takes lowercase flipkart as default bean id.
the metadata that prepared beacuase of annotation will update xml metadata by using beanfactorypostprocessor.



Vehicle.java:

```
package com.nt.beans;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class Viechle{
    @Autowired
     private Engine engg;
     //toString
    @Override
    public String toString() {
            return "Viechle [engg=" + engg + "]";
    }
            }
```

Engine.java:

```
package com.nt.beans;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
@Component("eng")
public class Engine{
    @Value("1001")
     private int id;
    @Value("DDIS")
     private String type;
     //setters
    public void setId(int id) {
            this.id = id;
```

```
    }
    public void setType(String type) {
            this.type = type;
    }
      //toString()
    @Override
    public String toString() {
            return "Engine [id=" + id + ", type=" + type + "]";
    }
            }
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<context:component-scan base-package="com.nt.beans"/>
</beans>
```

ClientApp.java:

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Viechle;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //getBean
            Viechle viechle=ctx.getBean("viechle",Viechle.class);
            System.out.println(viechle);
    }
}
```

@Value:

It is given to configure dependent values for simple properties of dependency injection.

@scope is given to specifie spring bean scope like singlton,prototype

@lazy is given to diasable preinstantiation of singlton scope bean and to enable lazy instantiation that means bean will be instantiated only when ctx.getbean() method is called.

```
Ex:     @componet("flipkart")
        @scope("singleton")
        @lazy
        public class Flipkart{
          --------
}
```

Note: to bring the effect of @lazy using xml tags we need to go for "lazy-init" attribute of bean tag

Java config annotations

@Componet, @Autowired and etc. are spring supplied annotations, if we use those annotations the code of spring bean becomes tightly coupled code to spring framework. This kills **non-invasive**features of spring.To overcome this problem use java config annotations given by JSE and JEE modules for these annotations functionality will be given by the containers and frameworks that we are using.

For Example:

For @Inject annotation spring provides it's functionalities and EJB provides it's functionalities. So if our code moves from 1 framework to another framework we can still continue with the same old java config annotations.

These annotations are part of JSE and J2EE specifications but implementation will be provided by different servers and frameworks. If we change runtime libraries of Application we can still use same annotations in the classes.

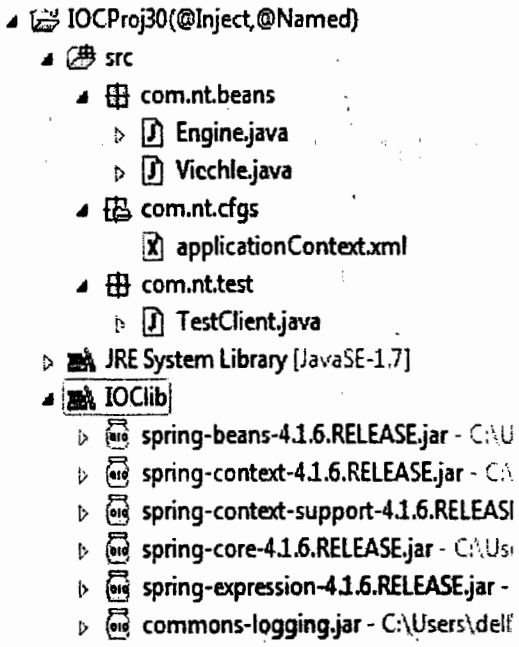
Ex:*@Inject, @Named, @Resource, @PostConstruct, @PostDestory* and etc..

@Inject:

It is given in JEE specification in javax.inject package having ability of performing byname/byType/constructor mode of autowiring.This is similar to *@Autowire* but the differences are

a)No required parameter in *@inject*

b)*@Inject* is java config annotation so non invasive spring application can be developed.



- **Add Spring-aop-<version>.jar+javcx.inject.jar (collect from internet).**

Note:-while working with spring application having both xml + annotations use xml file to configure predefined classes and use annotation to configure user defined classes . don't forget to add <context-component-scan> tag in xml file

while working with @autowired annotation if dependent bean is not found we can inject null value without rising exception by using required=false parameter. samething is not possible while working with @inject that means dependent bean is mandatory otherwise exception will be rised.

while working with @inject, @autowire anotation and if you want perform by name mode of autowiring only on one parameter when method are constructor contents more parameter then we need to use @named or @qualifier annotation at parameter level insdeed of at method or constructor level

Ex:-@inject

```
    public Flipkart(@Named("blueDart")Courier courier,Date dop){
    this.courier=courier;
    this.dop=dop;
}
```

Vehicle.java:

```
 package com.nt.beans;
import javax.annotation.Resource;
import javax.inject.Inject;
import javax.inject.Named;
@Named("viechle")
public class Viechle{
      private Engine engg;
     /*  @Inject
      public Viechle(Engine engg){
              this.engg=engg;
      }*/
      @Inject
      @Named("eng1")
      public void assign(Engine engg){
              this.engg=engg;
      }
             /* @Inject
      public void setEngg(Engine engg){
              this.engg=engg;
      }*/
      //toString
     @Override
     public String toString() {
             return "Viechle [engg=" + engg + "]";
     }
             }
```

Engine.java

```
package com.nt.beans;
import javax.inject.Named;
import org.springframework.beans.factory.annotation.Value;
@Named("eng")
public class Engine{
      @Value("1001")
       private int id;
      @Value("DDIS")
       private String type;
       //setters
     public void setId(int id) {
             this.id = id;
     }
     public void setType(String type) {
             this.type = type;
     }
        //toString()
     @Override
     public String toString() {
             return "Engine [id=" + id + ", type=" + type + "]";
     }
```

```
}
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns.xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<!-- <bean id="eng1" class="com.nt.beans.Engine">
<property name="id" value="1001"/>
<property name="type" value="DDIS"/>
</bean>
<bean id="eng2" class="com.nt.beans.Engine">
<property name="id" value="1002"/>
<property name="type" value="CRDI"/>
</bean> -->
<context:component-scan base-package="com.nt.beans"/>
</beans>
```

Client.java:

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Viechle;
public class ClientApp {
    public static void main(String[] args) {
        //create IOC Container
        ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //getBean
        Viechle viechle=ctx.getBean("viechle",viechle.class);
        System.out.println(viechle);
    }
}
```

@Inject can be applied at field/method (setter or orbitrary) method/constructor level. If **with *@Inject*, *@Named* kind of java config annotation we need to add 'javax.inject.jar' files.**

Total jarfiles:

IOCLab+Spring-aop-<version>.jar+javax.inject.jar

@Named:

It is a java config annotation of 'javax.inject.jar' (JSR-330 specification). This annotation can be used in 2 ways

1) To resolve ambiguity problems (when more than one bean of same type exists in a single container) (like *@Qualifier*) can be used with [illegible]

b)To configure/mark java classes as spring beans as alternate to spring specific annotations like *@Component* and etc...

eg:

Vehicle.java:

```
Package com.nt.beans;
@Named("vehicle")//bean configuration
Public class Vehicle{
@Inject
@Named("eng1")//to perform byName mode autowiring
//override toString()
.........
.........
}
```

Engine.java:

```
Package com.nt.beans;
Public class Engine{
Private int id;
Private String type;
//setters
........
........
//toString
}
```

applicationContext.xml:

```
<bean id="eng1" class="com.nt.beans.Engine">
...............
</bean>
<bean id="eng2" class="com.nt.beans.Engine">
...............
</bean>
<context:component-scan base-package="com.nt.beans"/>
```

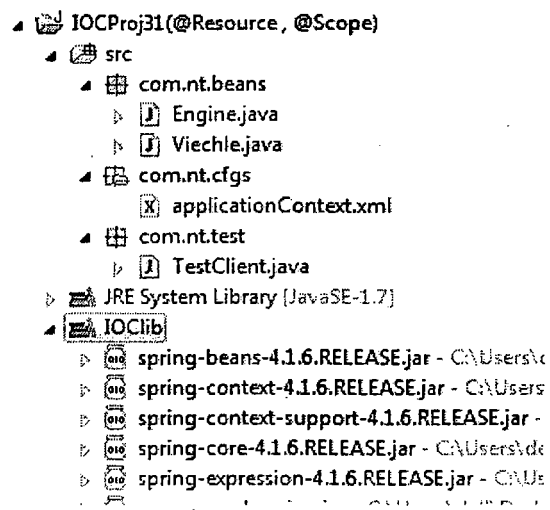
@Resource

a)It is jdk supplied java config annotations of javax.annotations pakage

b)Can be used to performe byType, byName mode of Autowiring.

c)Can be applied at field/method (setter/orbitrary) level.

d)By using name param of this annotation we can perform byName mode of autowiring.



Vehicle.java:

```
package com.nt.beans;
    import javax.annotation.Resource;
    import javax.inject.Named;
    import org.springframework.context.annotation.Scope;
    @Named("viechle")
    @Scope("prototype")
    public class Viechle{
        @Resource
          private Engine engg;
                public Viechle() {
               System.out.println("Viechle:0-param constructor");
        }
        /* @Resource
          public void assign(Engine engg){
                this.engg=engg;
          }*/
                //@Resource(name="eng1")
          public void setEngg(Engine engg){
                this.engg=engg;
          }
          //toString
        @Override
        public String toString() {
               return "Viechle [engg=" + engg + "]";
        }
               }
```

Engine.java:

```
package com.nt.beans;
import javax.inject.Named;
import org.springframework.beans.factory.annotation.Value;
public class Engine{
        private int id;
        private String type;
        //setters
    public void setId(int id) {
            this.id = id;
    }
    public void setType(String type) {
            this.type = type;
    }
       //toString()
    @Override
    public String toString() {
            return "Engine [id=" + id + ", type=" + type + "]";
    }}
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean id="eng1" class="com.nt.beans.Engine">
<property name="id" value="1001"/>
<property name="type" value="DDIS"/>
</bean>
<context:component-scan base-package="com.nt.beans"/>
</beans>
```

Client.java:

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.beans.Viechle;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC Container
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //getBean
            Viechle viechle=ctx.getBean("viechle",Viechle.class);
            System.out.println(viechle);
            ctx.getBean("viechle");
            ctx.getBean("viechle");
```

```
    }
}
```

Difference between @Autowired, @Inject and @Resource:

@Autowired	***@Inject***	***@Resource***
Spring Annotation	~~JEE~~ java configuration	JDK java config annotation
SupportsbyType, byName,constructor mode of autowiring	same	Supports byType, byname mode of autowiring.
Can be applied on field, settor,arbitory method Method,constructor	same	can be applied on field, settor method,arbitory method method.
Makes spring class as invasive	non-invasive class	non-invasive class
Param required is available	non-param	param name is available.
Use @Qualifier or @Name to problem. avoid ambiguty problem resolve ambiguity problem.	same attribute @Named to	We can use 'name'resolve ambiguity problem

@auif dependent is not found we can avoid exception by taking required=false.
if dependent is not available exception is rised
this annotation can be applied on arbitory method when it is having one or more params

@inject same
this annotation must applied only on one param arbitary method

@Scope

This is spring annotation which can be applied on class level to specify springbean scope.

```
@Named("vehicle")
@Scope("Singleton")
public class vehicle
{
.....
.....
}
```

If the annotation param name is "value" then we can place the context of that param without specifying paramname.

@Named(Value="Vehicle") is equal to *@Named ("vehicle")*

Note: As of now spring is providing functionality only to few java config annotations.

@PostConstruct, @PreDestory

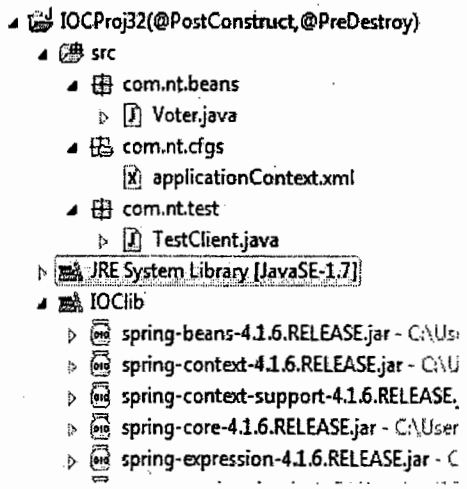
<u>Bean lifecycle configuration can be done in 3 ways:</u>

a)**Declarative Approach**
(By using init-method, destroy-method attribute of <bean> tag)
b)**Programatic Approach**
(By implementing InitializationBean, DisposibleBean interface)
c)**Annotation Approach**
(By using javaconfig annotations @PostConstruct, @PreDestroy)

@PostConstruct:
To configure normal method as custom init method
@PreDestroy:
To configure normal method as custom destroy method

Note: Both method should have the following signature

```
public void <method> (){
        -----
        -----
}
```



- ***Add javax.inject-<ver>.jar***
- ***Add spring-aop-<ver>.jar***

Voter.java:

```
package com.nt.beans;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.inject.Named;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Scope;
@Named("vt")
@Scope("singleton")
public class Voter {
    @Value("raja")
    private String name;
    @Value("30")
    private int age;
    @PostConstruct
    public void myInit(){
            System.out.println("Voter:myInit()");
            if(name==null || age<=0){
                    throw new IllegalArgumentException(" name,age must be set");
            }
    }
   @PreDestroy
    public void myDestroy(){
      System.out.println("Voter:myDestroy()");
       name=null;
       age=0;
    }
   public String checkVotingElgibility(){
       if(age<18)
            return name+" u r not elgible to vote";
       else
            return name+"u  r elgible to vote";
   }
}
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context.xsd">
   <context:component-scan base-package="com.nt.beans"/>
   </beans>
```

Client.java

```
   package test;
   import org.springframework.context.ApplicationContext;
   import org.springframework.context.support.FileSystemXmlApplicationContext;
   import com.nt.beans.Voter;
   public class ClientApp {
      public static void main(String[] args) {
            // create IOC Container
            ApplicationContext ctx =new
      FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //get Bean
            Voter vt=ctx.getBean("vt",Voter.class);
            String status=vt.checkVotingElgibility();
            System.out.println("Result::"+status);
            ((FileSystemXmlApplicationContext)ctx).close();
                  }}
```

Working with properties file in annotations environment

a) To detect the classes and to recognize them as spring beans through annotations we can use *<context:component-scan>*tag

b) To detect the specified 'properties' file that is configured through annotation we can use *<context:property-placeholder>*tag

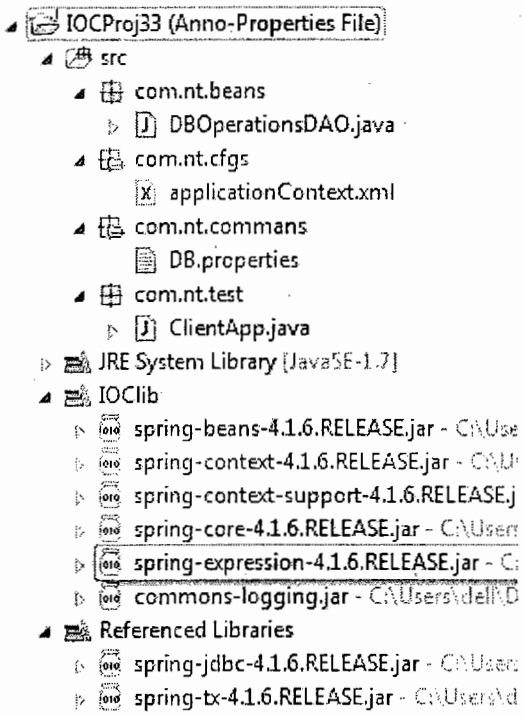
c) To inject the values of 'properties' files to "bean properties" we need to specify the keys of 'properties' file through placeholder **${<key>}.**

d) In <context:property-placeholder location=" ----- "/> tag we can specify multiple properties files locations as the (,) separation in location attribute.

e) <context:property-placeholder /> tag internally uses PropertyPlaceholderConfigurer" class to locate the properties file and to recognize placeholders (${---}).

f) PropertyPlaceholderConfigurer class not only looks to get given property value from properties file and it can also get it from java system properties

```
@Value("${java.v4.specification.vendor}")
 private string vendor;
```



DBoperations.java:

```
package com.nt.beans;
import javax.inject.Named;
import org.springframework.beans.factory.annotation.Value;
@Named("dao")
public class DBOperationsDAO {
    @Value("${jdbc.driver}")
    private String driver;
    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.user}")
    private String username;
    @Value("${jdbc.pwd}")
    private String password;
    @Override
    public String toString() {
            return "DBOperationsDAO [driver=" + driver + ", url=" + url
                    + ", username=" + username + ", password=" + password + "]";
    }
}
```

149page: while converting xml spring example application into xml +annotation approach example
congfigure all predefine classes using xml file and also configure user define classes using annotation
*while converting xml spring example application to 100%code based spring application configure predefine classes in the bean configure class in the method that are annotated with @bean and configure user define classes through annotations.
* while working with annotation + xml approach configure properties file by using context-property:placeholder tag similarly in 100% code approach configure by using @property source and also inject all the properties to envernment object of bean configurator class and get value from envernment object where ever required.
sample cod:

```
@configuration
@ComponentScan(----)
@PropertySource(value="classpath:com/nt/commons/jdbc.properties")
public class BeanConfigurator{
@Autowired
private Environment env;
}
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<context:component-scan base-package="com.nt.beans"/>
<context:property-placeholder location="src/com/nt/commons/DB.properties"/></beans>
```

DB.properties

```
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url =jdbc:oracle:thin:@localhost:1521:xe
jdbc.user=scott
jdbc.pwd=tiger
```

ClientApp.java:

```
package test;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.support.FileSystemXmlApplicationContext;
    import com.nt.beans.DBOperationsDAO;
    public class ClientApp {
        public static void main(String[] args) {
                //Create IOC Container
    ApplicationContext ctx =new
        FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //get Bean
                DBOperationsDAO dao=ctx.getBean("dao",DBOperationsDAO.class);
                System.out.println(dao);
        }
    }
```

Note: working with *<context: property-placeholder....>* tag is equal to working with 'PropertyPlaceholderConfigurer' Bean class of Spring API.

Developing spring core module application without xml file/(100% java code based spring core module app):

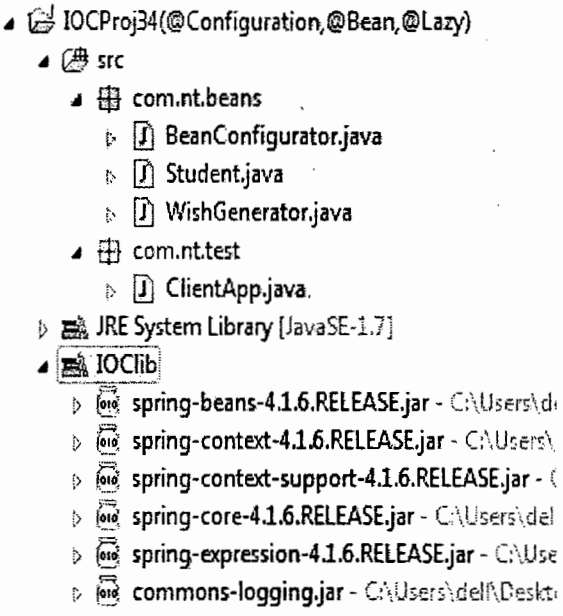
For this **we need to use "AnnotationConfigApplicationContext" to create core IOCcontainer by specifying "Configuration" class**. The java class that contains methods to create and return bean class objects is called "Configuration" class. We use ***@Configuration*** to configure ordinary class as configuration class and we use ***@Bean*** on the top the configuration class method which produces objects (to configure those object as bean object use *@Bean*).

Note:

1) 100% java code approach improves the performance because of no XML file configuration and also improves the readability.

2) We can take configuration class that is annotated with @Configuration as alternate to spring bean configuration file.

3) We place @Bean annotated methods in the configuration class to create and return predefined class objects and to make them as spring beans. But we can configure user defined classes as spring beans by using @Named, @Component annotations. **We link them to configuration class by using @ComponentScan annotation.**



BeanConfigurator.java:

```
package com.nt.beans;
    import java.util.Date;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    import org.springframework.context.annotation.DependsOn;
    import org.springframework.context.annotation.Lazy;
    import org.springframework.context.annotation.Scope;
    @Configuration
    public class BeanConfigurator {
        public BeanConfigurator() {
          System.out.println("BeanConfigurator: 0-param constructor");
        }
        @Bean(name="st")
        @Lazy(value=true)
        @Scope("prototype")
        public Student  createStudent(){
                System.out.println("createStudent()");
                return new Student();
        }
        @Bean
        @DependsOn("st")
        public Date  createDate(){
                System.out.println("createDate()");
                return new Date();
```

```
    }
    @Bean
    public WishGenerator  createWishGenerator(){
            System.out.println("createWishGenerator()");
            return new WishGenerator();
    }
```

Student.java:

```
package com.nt.beans;
public class Student {
    public Student() {
      System.out.println("Student:0-param constructor");
    }
            @Override
    public String toString() {

            return "101 --> raja";
    }
}
```

WishGenerator.java:

```
package com.nt.beans;
public class WishGenerator {
            public WishGenerator() {
      System.out.println("WishGenerator:0-param constructor");
    }
    @Override
    public String toString() {
            return "Good Morning";
            }}
```

ClientApp.java:

```
package test;
import java.util.Date;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.nt.beans.BeanConfigurator;
import com.nt.beans.Student;
import com.nt.beans.WishGenerator;
public class ClientApp {
    public static void main(String[] args) {
            //Create IOC Container
            ApplicationContext ctx=new
    AnnotationConfigApplicationContext(BeanConfigurator.class);
    //getBeans
    /*      WishGenerator wg=ctx.getBean(WishGenerator.class);
```

```
System.out.println(wg);
Student  st=ctx.getBean("st",Student.class);
System.out.println(st);
ctx.getBean("st");
                Date d=ctx.getBean(Date.class);
System.out.println(d); */
        } }  //main& Class
```

Limitations with above approach:

a) Kills non-invasive behavior of SpringBean
b) Kills loosly coupling behavior of spring bean.
c) Configuring pre-defined, third party supplied classes as spring beans is very complex.

Note:

The class that is specified with *@Configuration* is called configuration class. In that class we add *@Bean* on the methods that produce bean class objects.These methods will be executed automatically the moment IOC container started.

d) All the methods of configuration class that are annotated with *@Bean* will be executed automatically, the moment IOC container is created by specifying configuration class.

e) If you want to delay certain method execution until *ctx.bean(-)* is called, then use *@Lazy* annotation

f) In order to create Bean class object only after creating another bean class object then we can go for *@DependsOn* annotation.(in xml configurations we can use depends-on attribute of the <bean> tag for the same).

eg

```
@Configuration
Public class Beanconfiguration{
@Bean(name="wish")
@Lazy(value=true)
@dependsOn("dt")
public WishGenerator createWishGenerator(){
return new WishGenerator();
}
@Bean(name="dt")
@Scope("Singleto")
public Date createDate(){
return new Date();
}
}
```

Note:

Bean Instantiation takes place when IOC container is created is called "Eager Instantiation".
Bean Instantiation that takes place when *ctx.getBean()* is invoked is called Lazy instantiation.

Even though bean scope is singleton if you want to delay bean instantiation until getBean(-,-) is called in ApplicationContext container use @Lazy annotation/Lazy="true" of <bean> in xml.
While working in 100% code based spring application we used to take configurator class specifying @ComponentScan even though all beans are user defined classes.

In 100% code based spring application development we can't use PropertyPlaceholderConfigurer to recognize and use placeholders in configuration class itself. But it will recognize and use placeholders specified in spring beans in the @Value annotation.

Spring JDBC/DAO

Where should we use jdbc and where should we use hibernate?

Ans: While working with jdbc if you want to process huge amount of records then we need to deal with single jdbc ResultSet object.While working with hibernate, if we want to process huge amount of records then we need to take multiple objects representing multiple records. This is a problem and it may crash the system, while creating huge amount of objects.

Use jdbc for offline applications that deals with huge amount of data where we don't need database portability feature. (changing DB software without modifying the code).

eg: Storing and managing sensus details, political party membership details by performing data validation and also correction.

While developing online application (Internet applicaions) that deals with limited amount of personalized data then use Hibernate.

eg: Online shopping mobile recharge online etc...

While developing internet application that performs batch processing of huge amount of data then use jdbc technology.

eg: Courier Application that takes all orders and process at a time.
eg: mobile number activation application that are activates set of numbers at a time.

Spring jdbc provides abstraction layer on plain JDBC and simplifies persistence logic development.Instead of using plain JDBC in offline applications, Intranet applications that deals with huge amount of data through batch processing it is recommended to use spring JDBC.

Irrespective of whether application is offline/online use jdbc/spring jdbc to process huge amount of data that comes batch by batch and similarly use hibernate to process limited amount of data.

Plain JDBC code:

- Register JDBC driver
- Establish the connection
- Create statement
- Send and execute SQL query
- Gather Results and process Results
- Exception handling
- Close JDBC objects

Here we need to write both common logics and applications specific logics in the application, so it raises boiler plate code problem.

Limitations with plain JDBC:

- Boiler plate code problem
- Exception handling is mandatory
- Processing Resultset to get result in desired type is complex process.
- No support for proper transaction management.

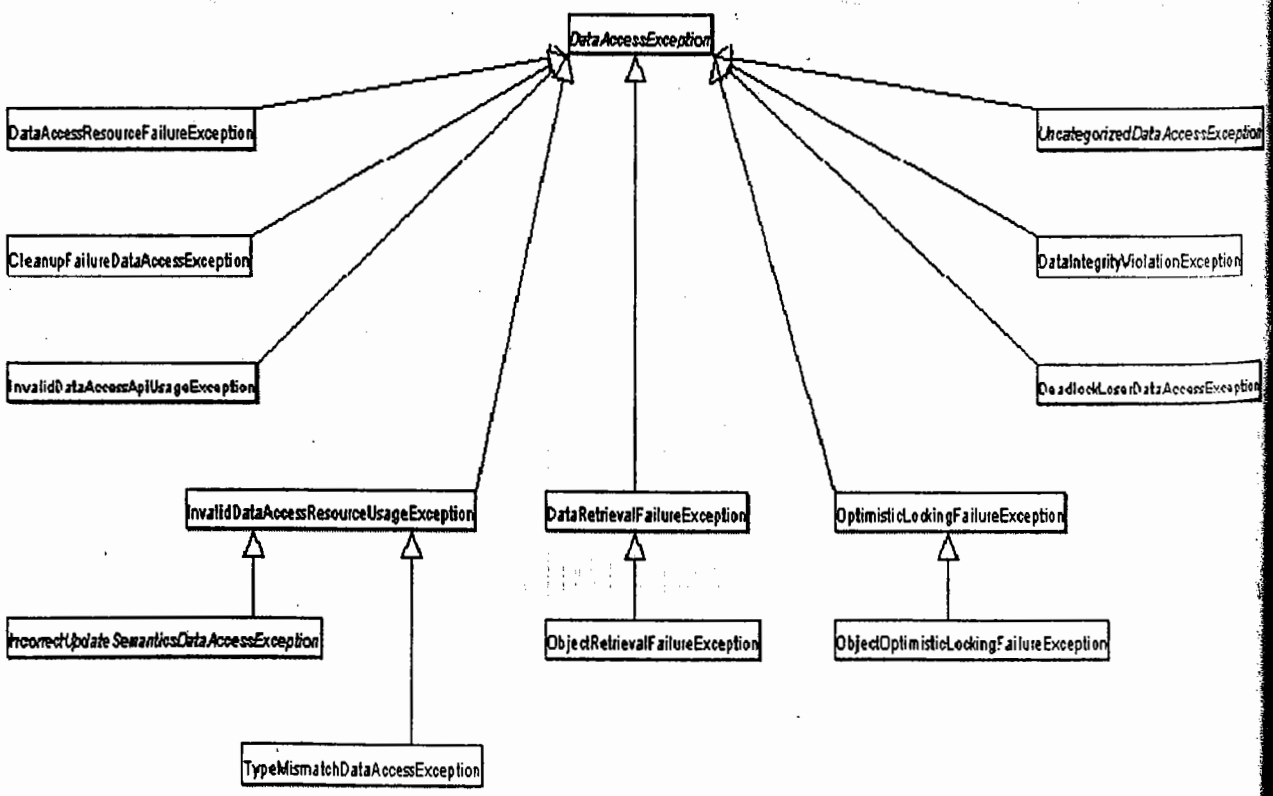
Spring JDBC App:

- Configure and get JDBC Template class object
- Send and execute SQL query.
- Gather results and process results

Note:

Here we need to take care of only application specific logics development.

Spring JDBC/DAO Advanatges:



- Solves Boiler plate code problem
- Makes programmer to take care of only application specific logics as shown above.
- Raises unchecked Exceptions (Sub classes of DataAccessException or DataAccessException class itself) so the exception is optional to catch and handle. It internally uses ***exception rethrowing concepts*** to convert checkedException (SQLException) into unchecked exception.
- Gives various methods to get the records of ResultSet in desired types.
- Takes care of opening connection and close connection activities.
- Support both Named and positional params in SQL query.

provides callback interfaces to write customized logics to get customized results.
gives various direct methods like queryForXxx(-,-) and query(-,-) to get the results.

Allows to use callback interface to develop more customized persistence logic by using plain JDBC code.

Action	Programmer	Spring JDBC
Configure DataSource	Yes	
Creating/getting connection		Yes
Creating Statement		Yes
Preparing SQL query with Params	Yes	
Sending executing SQL query in Database software	Yes	
Iterating through ResultSet (if needed)		Yes
Exception Handling		Yes
Transaction management		Yes
Closing JDBC objects		Yes

while working with spring jdbc all the common logics are taken care by springjdbc itself so we can avoid boiler plate code problem .

Spring JDBC api pkgs

org.springframework.~~JDBC~~ jdbc and its sub pkgs
org.springframework.dao and its sub pkgs
(this package contains only Exceptions)

Note: Spring JDBC/ORM module internally uses multiple other technologies/Frameworks to develop persistence logics but translate the exception of underying technology/framework to spring's DataAccessException hierarchy.

There are 5 methodlogies/approaches to develop persistence logic in spring JDBC.

1) **JdbcTemplate**
2) **NamedParameterJdbcTemplate**

3) **SimpleJdbcTemplate (removed spring 4.0)**
4) **SimpleJdbcInsert, SimpleJdbcCall**
5) **Modeling JDBC operations as Java objects(Mapping SQL operations as subclass using SQLQuery, SqlUpdate classes. (JDO Style))**.

While working with any technique of spring JDBC we need to use JDBC connection to interact with database software.It is recommended to use JDBC connection pool based JDBC connection instead of direct JDBC connection.Do not use spring's built-in classes like *DriverManagerDataSource, SingletonConnectionDataSource* and etc. to work with spring JDBC connection pool because those classes actually doesn't pool the connection objects more ever creates JDBC connection object on demand.

What is the JDBC connection pool that should be used in our spring application?
Ans:

1) If our spring api is standalone app running outside the server then use third party JDBC connection pool like *c3po* or *apacheDBCP* and etc..
2) If our spring app is deployable in server then use JDBC connection pool

Using JdbcTemplate

This is central class of spring JDBC provides abstraction layer on plain JDBC and simplifies persistence logic development.This class takes care of common workflow activities of JDBC programming like opening connection, creating statement, closing connectin and etc.... but makes the **programmer to just provide SQL query and to extract results**.This class translates JDBC generated *SQLExceptions* into spring specific unchecked exception like *DataAccessException.*

This allows programmer to avoid common errors in application development like forgetting to close JDBC objects.To create JdbcTemplate class object we need JDBC DataSource object.We generally configure JdbcTemplate in spring bean configure file and use inject that object to DAO class.

persistence –beans.xml:

```
<bean id="drds" class="org.springframework.jdbc.dataSource.DriverManagerDataSource">
.........................
.........................
</bean>
<bean id="template" class="org.springframework.jdbc.Support.jdbcTemplate">
<property name="dataSource" ref="drds"/>
</bean>
```

There are 2 approaches of working with JdbcTemplate
1) Using direct methods (No need of JDBC code)
2) Using callback interfaces (Need to write plain JDBC code to place customized persistence logic)

Using direct method:
Working with direct methods of JdbcTemplate class.
Select-Operations: (query, queryForXxx(-) methods)
queryForInt(), queryForLong()

To execute query that gives single numeric value as result.
eg: Selecy count(*) from student;

queryForMap():
To execute the query that gives single record.
eg: Select * from student where sno=1001;

queryForObject()
To execute the query and to get result into predefined or Bo class object.
eg: Select * from student where sno=1001;

queryForList()
To execute query that gives multiple records.
eg: Select * from student;

queryForRowSet()
To execute query that gives JDBC RowSet object and etc...

Note: If the above methods are called with one argument they internally uses JDBC simple statements object.

Note: If the above methods are called with 2 or 3 arguments the internally uses JDBC *PreparedStatement* object.

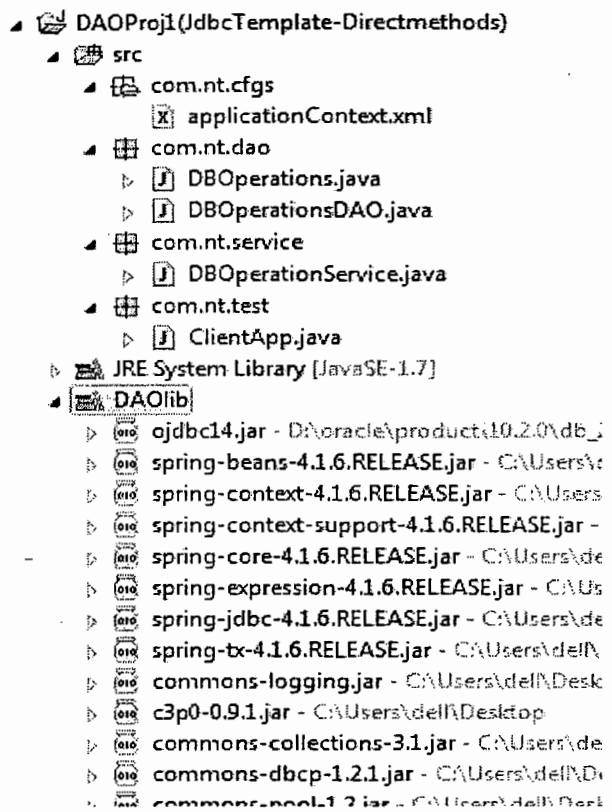
Non-Select operations
Update (-) method

batch updation
batchUpdate (-) and many more

Note: Sprind DAO internally uses Exception rethrowing concept to translate underlying Jdbc persistence technology generated SQLException to DataAccessExceptions.

eg:-

```
public int update(String query, String ... args){

try{
------//Jdbc code
------
}catch(SQLIntrigityConstraintVoilationException scve){
throw new DuplicateKeyException();  exception rethrowing
} catch(SQLException se){
throw new DataAccessException();    exception rethrowing
    }
}
```



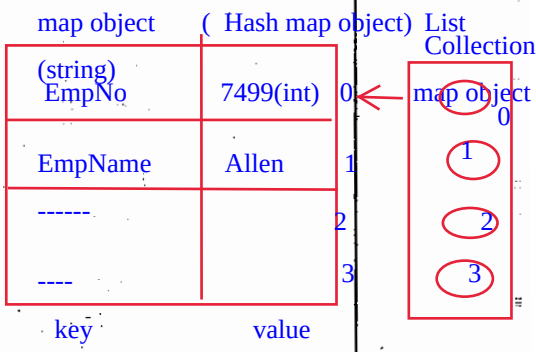
DBoperations.java

```
package com.nt.dao;
import java.util.List;
import java.util.Map;
public interface DBOperations {
    public int  insert(int no,String name,String job,long sal);
    public int  getSalary(int no);
    public Map<String,Object> listEmpDetails(int no);
    public List<Map<String,Object>> listAllEmpDetails();
    public int UpdateEmpSalary(int no,int newsal);
    public int deleteEmp(int no);
}
```

DBOperationsDAO.java

```
package com.nt.dao;
import java.util.List;
import java.util.Map;
import org.springframework.jdbc.core.JdbcTemplate;
public class DBOperationsDAO implements DBOperations {
    private static final  String  INSERT_QRY="insert into emp(empno,ename,job sal)
    values(?,?,?,?)";
    private static final  String  GET_EMP_SALARY="select sal from emp where empno=?";
    private static final  String GET_EMP_DETAILS_BY_NO="select empno,ename,job,sal
from    emp where empno=?";
    private static final String  GET_ALL_EMP_DETAILS="select empno,ename,job,sal from
Emp";
    private static final String  UPDATE_EMP_SALARY="update emp set  sal=? where
```

```
empno=?";
        private static final String  DELETE_EMP="delete from emp where empno=?";
                private JdbcTemplate jt;
                  public void setJt(JdbcTemplate jt) {
                        this.jt = jt;
                }
        @Override
        public int insert(int no, String name, String job, long sal) {
                int result=jt.update(INSERT_QRY, no,name,job,sal);
                return result;
        }
        @Override
        public int getSalary(int no) {
         int salary=jt.queryForInt(GET_EMP_SALARY,no);
                return salary;
        }
        @Override
        public Map<String,Object> listEmpDetails(int no) {
          Map<String,Object> map=jt.queryForMap(GET_EMP_DETAILS_BY_NO,no);
                return map;
        }
        @Override
        public List<Map<String,Object>> listAllEmpDetails() {
                List<Map<String,Object>> list=jt.queryForList(GET_ALL_EMP_DETAILS);

                return list;
        }
        @Override
        public int updateEmpSalary(int no, int newsal) {
                int result=jt.update(UPDATE_EMP_SALARY,newsal,no);
                return result;
        }
        @Override
        public int deleteEmp(int no) {
         int result=jt.update(DELETE_EMP,no);
                return result; }}
```



DBOperationService.java

```
package com.nt.service;
import java.util.List;
import java.util.Map;
import com.nt.dao.DBOperationsDAO;
public class DBOperationService {
                private DBOperationsDAO dao;
        public void setDao(DBOperationsDAO dao) {
                this.dao = dao;
        }
                public String registerEmp(int no,String name,String job,int sal)
        {
```


:-DAO class should contain only persistence logic you should not small-2 arithmetic logical operations like if conditions, c=a+b etc

```
        int result=dao.insert(no, name, job, sal);
        if(result==0)
                return no+" Emp is not registered";
        else
                return no+" Emp is registered";
}
        public int showSalary(int no){
        int salary=dao.getSalary(no);
        return salary;
}
        public String hikeSalary(int no){
         //write B.logic
        //get Salary
        int salary=showSalary(no);
        int newsal=0;
        if(salary>=50000)
                newsal=Math.round(salary+(salary*0.1f));
        else
                newsal=Math.round(salary+(salary*0.2f));
                //use DAO
        int result=dao.updateEmpSalary(no, newsal);
         if(result==0)
           return no+" Emp Salary not hiked";
         else
                 return no+"Emp Salary  hiked";
}
public  Map<String,Object>  getEmpDetails(int no)throws Exception{
        Map<String,Object> map=dao.listEmpDetails(no);
        if(map!=null)
                return map;
        else
           throw new Exception(no+" Emp not found");
}
        public  List<Map<String,Object>> getAllEmpDetails()throws Exception{
          List<Map<String,Object>> list=dao.listAllEmpDetails();
          if(list!=null)
             return list;
          else
                  throw new Exception("Employees not found");
}
public  String fireEmp(int eno){
        int result=dao.deleteEmp(eno);
        if(result==0)
                return eno+" Employee not found";
        else
                return "Employee deleted";
}}
```

Note: In plain jdbc when we execute select query we get result set object which is not a serialazable object to send over the network but In springJdbc query for Xxx(---) method like query for map or query for list gives the results directly in the form of collection which are serializable object to send over the network.
2: while working with standalone spring applications we cn use appache dpcb or c3p0 standalone connection pool but it recommnded to use appache dpcb when compare to c3p0 for better performance.

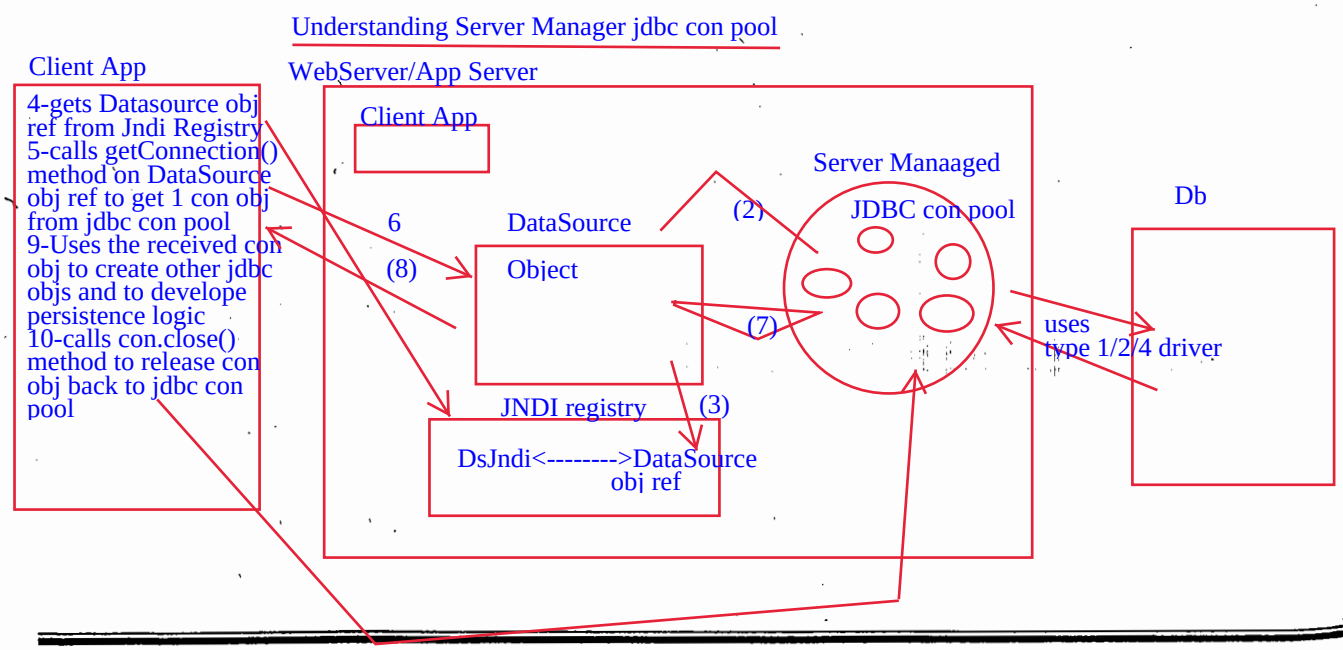
applicationContext.xml

```
<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.3.0.xsd">
<!--  <bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean> -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
<property name="driverClass" value="oracle.jdbc.driver.OracleDriver"/>
<property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="user" value="scott"/>
<property name="password" value="tiger"/>
<property name="initialPoolSize" value="10"/>
<property name="maxPoolSize" value="20"/>
</bean>
<bean id="template" class="org.springframework.JDBC.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="DBODao"  class="com.nt.dao.DBOperationsDAO">
<property name="jt" ref="template"/>
</bean>
<bean id="DBOService"  class="com.nt.service.DBOperationService">
<property name="dao" ref="DBODao"/>
</bean>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.DBOperationService;
public class ClientApp {
    public static void main(String[] args) throws Exception{
        //Create IOC Container
        ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //get Service Bean
        DBOperationService service=(DBOperationService)ctx.getBean("DBOService");
        // Call B.methods
        System.out.println("7499 Emp Details"+service.getEmpDetails(7499));
        System.out.println("All Emp Details"+service.getAllEmpDetails());
        System.out.println("7499 Emp Salary(Before hike)"+service.showSalary(7499));
        System.out.println("7499 Emp Salary hiking..."+service.hikeSalary(7499));
        System.out.println("7499 Emp Salary(After hike)"+service.showSalary(7499));
        //System.out.println(service.registerEmp(9001,"Raja","progammer",8999));
        //System.out.println("Fire Emp"+service.fireEmp(9001));
    }//main
}//class
```

Jar files: IOCLib jar files+object 14/6.jar+org.springframework.JDBC<ver>.jar Commons-pool.jar+commons-dbcp.jar+org.springframework.+n<ver>.jar (from spring 2.5 software) or c3o-0.9.1.2.jar (To work with c3po pool)



client can be a inside the sever like (servlet component) or outside the server like (standalone application)
in a project only team leader or project leader creates jdbc connection pool and remaining members will use that connection pool

JNDI Registry and ServerManaged Jdbc connection pool

To provide global visibility to object or object reference it will be placed in jndi registry having nickname/jdni name.Every webserver/Application server gives one built-in jndi registry.In server managed JDBC connection pool envinorment the JDBC con pool will be represented by *DataSource* object and its reference will be placed in jndi registry of server for global visibility having jndi name the apps deployed in server or running from outside server can access *DataSource* object reference from jndi registry to get JDBC connection Object from JDBC connection Pool.In one server we can create multiple JDBC connrction pool belonging to same on diff database softwares.All connection object in connection pool represents connectivity with same database software.The client application that wants to use connection pool can be there outside the server as standalone application and can be there inside the server as deployable servlet component, jsp component, Ejb component.

With respect to diagram:

TL or PL or Admin makes server to use typ1/2/4 JDBC driver to interact with database software and to create servermanaged JDBC connection pool having connection objects.TL/PL/Admin creates DataSource object representing JDBC connection pool places that *DataSource* object reference in jndi registry having jndi name for global visibility.

In weblogic softwarewe can create multiple domains and each domain acts as 1 domain server.Multiple projects of a company uses multiple domains servers of weblogics software ~~1~~ on 1 per project basis i.e the weblogic software will be installed only for 1 ~~in~~ time in a common computer and in that software multiple domains will be created for multiple projects on 1 per project basis.

Procedure to create user-defined domains server in weblogic 12c:

Start---->Oracle----->oracle home------>weblogic server12c------>tools---->conf-wizard---->create new domain (domain location:..... sp27Domian)------>next---------->next

Name: javaboss

Pass: javaboss1

Confo-pass:javaboss1

Next--->next--->select admin server---->next--->proj:7080----->create--->next---------->finish.

Procedure to create JDBC DataSource pointing JDBC con pool for oracle in "sp~~27~~86Domain" server of weblogic:

start sp27Domain server

D:\weblogic 12c\user-project\Domains\sp27Domain\Startweblogic.cmd

Open admin console of that domain server

http://localhost:7080/console------>

username: javaboss

password: javanoss1

login

weblogic-

type: app Server
vendor:BEA systems/Oracle corp
version:12c(compatible with jdk 1.7)
Commercial
default port:7001
Allows to domains

Create a JDBC DataSource

Adminconsole--->services----->datasources--->new---->generic datasource

Name ds1

Jndi name: dsjndi

DatabaseType: Oracle

Driver: Oracle's Driver (thin) for service connection

Next--->next
Database name:xe
Hostname : localhost
Port: 1521
Db username: Scott
Password:tiger
Conform pass: tiger ------>Test Configuration---->next
Select admin Server------>Finish
Launch
Connection pool tab--->Advanced---->
Min:3
Max: 30
Save
Advanced:
Shrink frequency : 900 (specifies the idel connection onjects of JDBC connection pool will be destroyed automatically for every 900sec)
------>save

Spring gives 1 built-in *FactoryBean* class to get object from jndi registry based on given jndi name.
This FactoryBean class name is*org.springframework.jndi.JndiObjectFactoryBean*

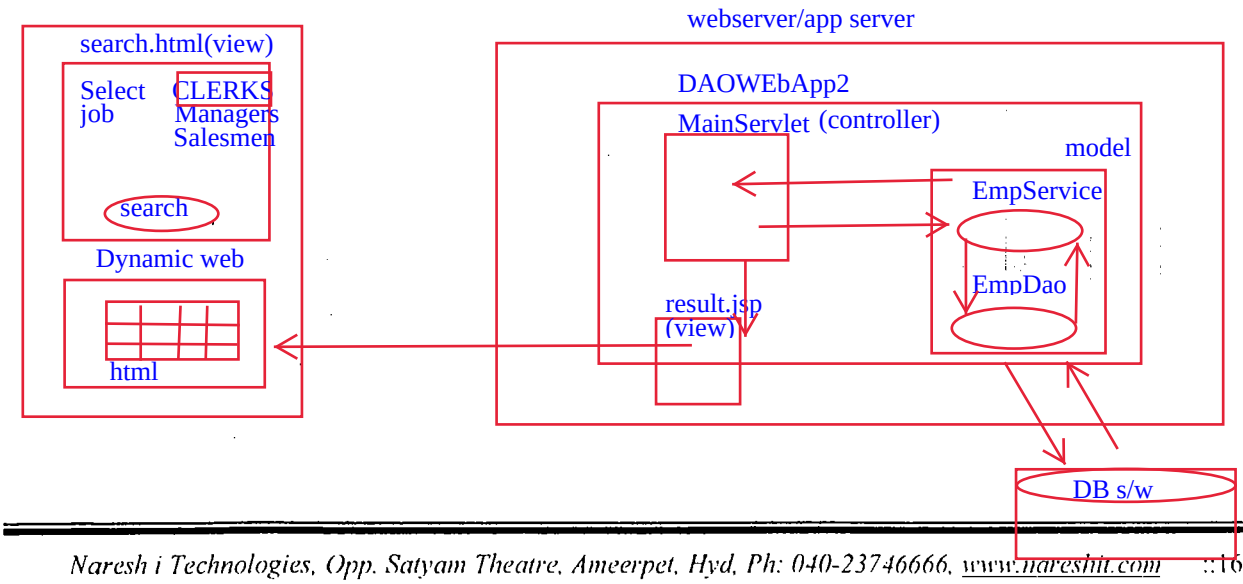
eg: in bean Configuration file.

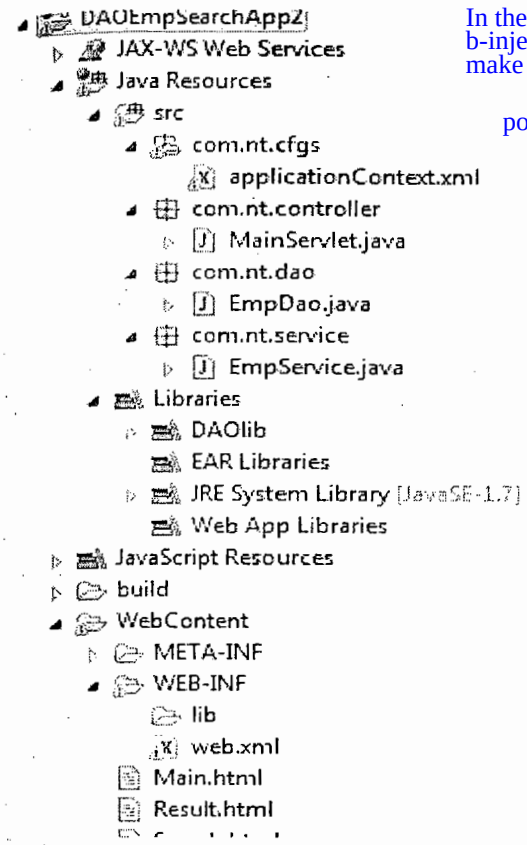
```
<bean id="joFb" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="JndiName" value="DSJndi"/>
</bean>
<bean id="template" class="org.springframework.JDBC.core.JdbcTemplate"
<property name="dataSource" ref="joFb"/>
</bean>
```

jdbc

Since the above bean is *FactoryBean* the gathered *DataSource* object will be injected here.

Web Application by using mvc2,DAO design patters:
The below application is web application so it is deployed in server so use server managed jdbc con pool.





In the above application a- is used server managed jdbc application
b-inject dao class object to service class object
make controller servlet create ioc container & to get service class object

pom.xml:-

EmpDAO.java

```
package com.nt.dao;
import java.util.List;
import java.util.Map;
import org.springframework.jdbc.core.JdbcTemplate;
public class EmpDao {
    private JdbcTemplate jt;
    public void setJt(JdbcTemplate jt) {
            this.jt = jt;
    }
    public List getEmpDetails(String condition){
            List<Map<String,Object>> list=jt.queryForList("select empno,ename,job,sal
from emp    where job in "+condition, new Object[]{});
            return list;
    }
}
```

EmpService.java

```
package com.nt.service;
import java.util.List;
import java.util.Map;
import com.nt.dao.EmpDao;
public class EmpService {
    private EmpDao dao;

    public void setDao(EmpDao dao) {
```

```
            this.dao = dao;
        }

        public List search(String job[]){
            //prepare condition
            StringBuffer sb=new StringBuffer("(");
            for(int i=0;i<job.length;++i){
                if(i==job.length-1){
                    sb.append("'"+job[i]+"'");
                }
                else{
                    sb.append("'"+job[i]+"',");
                }
            }//for
            sb.append(")");
            String condition=sb.toString();
            //Use DAO class persistence logic
            List<Map<String,Object>> list=dao.getEmpDetails(condition);
            return list;
        }//search
    }//class
```

MainServlet.java

```
package com.nt.controller;
    import java.io.IOException;
    import java.util.List;
    import java.util.Map;
    import javax.servlet.RequestDispatcher;
    import javax.servlet.ServletException;
    import javax.servlet.http.HttpServlet;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.support.ClassPathXmlApplicationContext;
    import com.nt.service.EmpService;
    public class MainServlet extends HttpServlet {
        ApplicationContext ctx;
        EmpService service;
        public void init(){
            //create IOC container
            ctx=new
        ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            // get Service class bean
            service=(EmpService)ctx.getBean("service");
        }
            @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                throws ServletException, IOException {
            //read form data
```

```
            String jobs[]=req.getParameterValues("job");
            //use Service class
            List<Map<String,Object>> list=service.search(jobs);
            // keep results in request attribute
            req.setAttribute("result",list);
            //forward the request to result page
            RequestDispatcher rd=req.getRequestDispatcher("/Result.jsp");
            rd.forward(req,resp);
    }//doGet(-,-)
            @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
            doGet(req,res);
    }//doPost(-,-)
}//class
```

applicationContext.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.3.0.xsd">

    <bean id="jofb"  class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName"  value="java:/comp/env/mypool"/>
    </bean>
    <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="jofb"/>
    </bean>
    <bean id="dao"  class="com.nt.dao.EmpDao">
    <property name="jt" ref="template"/>
    </bean>
    <bean id="service"  class="com.nt.service.EmpService">
    <property name="dao" ref="dao"/></bean></beans>
```

Mian.jsp

```
<frameset rows="40%,60%">
    <frame name="f1"  src="Search.jsp"/>
    <frame name="f2"/>
    </frameset>
```

Result.jsp

```
<%-- <%@page import="java.util.*" %> --%>
<%--  <%//read request attribute
  List<Map<String,Object>>
list=(List<Map<String,Object>>)request.getAttribute("result");
   if(list!=null){ %> --%>
<!--     <table> -->
<!--      <tr> -->
<!--        <th>NO</th><th>NAMe</th><th>JOB</th><th>Salary</th>  -->
<!--      </tr> -->
<%--   <% for(Map map:list){  %> --%>
<!--      <tr> -->
<%--       <td><%=map.get("EMPNO") %></td> --%>
<%--       <td><%=map.get("ENAME") %></td> --%>
<%--       <td><%=map.get("JOB") %></td> --%>
<%--       <td><%=map.get("SAL") %></td> --%>
<!--      </tr> -->
<%--  <%    } //for --%>
//   }//if
<%--    %> --%>
<!--   </table>  -->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:choose>
<c:when test="${not empty result}">
<table>
<tr>
<th>NO</th><th>NAMe</th><th>JOB</th><th>Salary</th>
</tr>
<c:forEach var="map" items="${result}">
<tr>
<td><c:out value="${map.EMPNO}"/></td>
<td><c:out value="${map.ENAME}"/></td>
<td><c:out value="${map.JOB}"/></td>
<td><c:out value="${map.SAL}"/></td>
</tr>
</c:forEach>
</table>
</c:when>
<c:otherwise>
    No Employee Found
</c:otherwise>
</c:choose>
```

Search.jsp

```
<h1><center>Searh EmpDetails </center></h1>
<form action="controller" target="f2">
 Select job(s)  <select name="job" multiple>
<option value="CLERK">CLERKS</option>
<option value="MANAGER">MANAGERS</option>
```

```
<option value="SALESMAN">SALESMEN</option>
<option value="ANALYST">ANALYSTS</option>
<option value="PRESIDENT">PRESIDENTS</option>
</select>
<input type="submit" value="search"/>
</form>
```

web.xml

```
<web-app>
<servlet>
<servlet-name>Main</servlet-name>
<servlet-class>com.nt.controller.MainServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Main</servlet-name>
<url-pattern>/controller</url-pattern>
</servlet-mapping>
</web-app>
```

Jar files: spJDBCLib jarfile+Ojdbc14.jar+JSTL.ajr+Servlet-api.jar+standard.jar----------->collect from tomcat installation (perform Deployment assembly)
While developing eclipse based web application the jar files placed in build path will not be move to *WEB-INF/lib* folder automatically. For that we need to perform deploymentAssemble separately.
Right click on project----->buildpath--->configure buildpath---->deployment assembly--->Add--->build path entries--->select jar files---->apply--->ok

To deploy Eclipse web project in Ntsp57 Domain of weblogic:

1- Start N+Sp27 domain server
<weblogic_name>\user-porjects\domains\N+Sp27domain\startweblogic.cmd
2- Export war file to N+Sp27 Doamin server
Right click project-->export-->warfile
Destination: D:\.... AutoDeploy/DAOEmpSearchApp2.war

3- Test the application.
http://localhost:7080/DAOEmpSearchApp2/main.html

Procedure to create user defined domain in Glasspringframeworkish server:
<Galsspringframeworkish_home>\bin>
Asadmin create-domain --adminport=4343 --user=testuser Ntsp~~27~~ 86

Enter password: testuser
Re-Enter password: testuser
.....
.....
Domain created
To install glasspringframeworkish 4.x extract the zip file
Glasspringframeworkish uses separate port no for http operations and separate portno for admin operations.

Procedure to create jdbc DataSource pointing to JDBC con pool for oracle in GNNtsp57 Domain server of Glassspringframeworkish:

1-- Place Ojdbc14/6.jar file in
<Glassspringframeworkish_home>/domains/GFNTSP57Domain/lib0/ext folder

2-- **Start the domain server**
<Galsspringframeworkish_home>\bin>asadmin start-domain NtSp~~57~~ GFNTSp86
Open admin console of domain server
http://localhost:~~46460~~ 4343
username: testuser
password: testuser
login

3- **For Stop domain**
<Glassspringframeworkish_home>\bin>asadmin Stop_domain Ntsp57

4- Create jdbc conn pool for Oracle
Admin console---->resource--->jdbc--->hdbcConnection pools--->new--->pool name: pool1
Reource type: javax.sql.DataSource
Database Drivervendor: Oracle----->Next
Initial and min pool size: 3
Max pool size:100
.....
.....
Additional properties
User:scott
DBName: Xp
Password:tiger
Servername:localhost
Url:jdbc:oracle:thin@localhost:1521.xe
Port no:1521
---->finish
Launch pool1--->ping
Create JDBC DataSource pointing to the above JDBC connection pool:
Adminconsole---->resource--->JDBC-->jdbcresources---->new--->jNDi name:DBjndi DSjndi
Pool name: pool1 ----->ok
To deploy application in Glassspringframeworkish Ntsp27Domain server of glassspringframeworkish place directory or war File in
<Glassspringframeworkish_home>\domain\Ntsp27\AutoDeployFolder

Note: While working with spring JDBC in servlet then application context constructor should be ClassPathXmlApplicationContext.
If it will FileSystemXmlAppContext. Then it will not work
Remember here src Folder should not mension in the file path of applicationContext.xml
Make that xsd file to beans-2.0.xsd

Creating JDBC connection pool for oracle in tomcat server:

Step1: make sure that tomcat-jdbc.jar file is available in <tomcat_home>/lib folder

Note: This jar file is builtin file in tomcat but must be arranged separately for other versions of tomcat by extracting apache-tomcat.jdbc-1.1.0.1_bin.zip file.

Step2: place ojdbc 14/6.jar file in *<tomcat_home>/lib folder*

Step 3: place the following *<Resource>*tag in *<tomcat_home>confg/context.xml*
File as the subtag of *<context>* tag

```
<Resource name="myPool" type="javax.sql.DataSource"
Factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
driverClassName="Oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:@localhost:1521:xe"
Username="Scott" password="tiger"  maxActive="20" maxIdle="10" validationQuery="SELECT
SYSDATE FROM DUAL" />
```

Step 4: restart the server

Note: To use the above JDBC connection pool in web application we need to place jndi name as shown below

Java:/comp/env/myPool

prefix Prefix (JNDI name)

Note: If you deploy any web application in tomcat server through eclipse IDE by configuration tomcat server with eclipse Ide. It is recommended to place "context.xml" file in servers folder of package explorer in order to use tomcat server managed JDBC connection pool in that webapplication.

Callback Interfaces

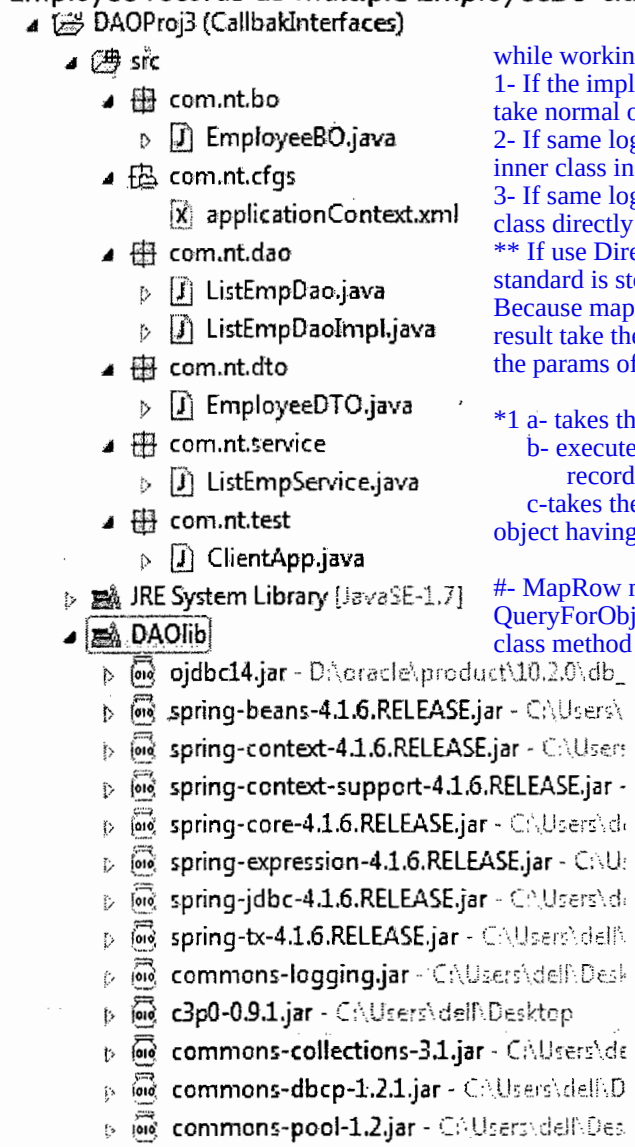
The method that is called by underlying jvm/container/server/etc.... automatically and dynamically is callback method.The interface that contains such methods is called callback interfaces.Spring provides lots of callback interfaces along with lots of templates classes. We can use these callback interfaces to write customized logics to process data and to get data in customized manner. spring jdbc callback method expose some jdbc object so that we can use these jdbc object to develop custom logics to get custom results like single record into BO object, collection of record into list BO object.

For example the *queryForMap(-)* gives single record in the form of map object having colum name as *key* and column *value* as the values in map dataStructure element. But industry standard is getting that record as *BO* class object. For this spring gives *RowMapper* call back interface.Various callback interfaces of spring JDBC are

StatementCallback--->Allows to work with statement object
PrepareStatementCallback
CallableStatementCallback
PrepareStatementCreator-->provides con object to create preparedStatement
PrepareStatementSetter-->provides preparedStatement object to set value to queryParams.
RwMapper---->To process 1 record of ResultSet at a time.
ResultSetExtractor---->To process multiple records of ResultSet at a time.

Note: The implementation class methods of callback interfaces will be called by *JdbcTemplate* internally.

eg: Application to get single employee record as *EmployeeBO* class object and multiple Employee records as multiple *EmployeeBO* class objects stored in *List* Collection.



while working with callback interfaces follow this suggestion:-
1- If the implementation logic of callback() method is required in multiple dao classes then take normal outer class as implementation class of callback interface.
2- If same logic is required only in multiple method of one DAO class then go for normal inner class inside dao class.
3- If same logic is required in only one method of dao class then prefer anonymous inner class directly or through lamda(->) expression.
** If use Direct() methods of jdbc template we get each record as map objects, But Industry standard is storing one record in BO object & multiple records into list of BO object. Because map objects holds column names as the key's unnecessary. so to get customized result take the support of callback interface & methods which exposes various jdbc object as the params of callback methods.

*1 a- takes the given query and sets the given parameter value.
b- executes the query and gets resultset object with 1 records and also gets index of the record in the table.
c-takes the given EmpRowMapper object(inner class oject) and calls maprow on that object having resultset object,index.

#- MapRow method converts the resultset object record to BO objects and gives to QueryForObject() method defination goes to DAO class method definition---goes to service class method definiton.

EmployeeBO.java

```
package com.nt.bo;
public class EmployeeBO {
    private int no;
    private String ename,job;
    private int sal;
    public EmployeeBO() {
    }
    public EmployeeBO(int no, String ename, String job, int sal) {
            this.no = no;
            this.ename = ename;
            this.job = job;
            this.sal=sal;
    }
    public int getNo() {
            return no;
    }
```

```
    public void setNo(int no) {
        this.no = no;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public String getJob() {
        return job;
    }
    public void setJob(String job) {
        this.job = job;
    }
    public int getSal() {
        return sal;
    }
    public void setSal(int sal) {
        this.sal = sal;
    }
}
```

ListEmpDAO.java

```
package com.nt.dao;
import java.util.List;
import com.nt.bo.EmployeeBO;
public interface ListEmpDao {
    public  EmployeeBO getEmpDetailsByNo(int eno);
    public List<EmployeeBO> getEmpDetailsByDesg(String desg);
}
```

ListEmoDAOImpl.java

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.jdbc.core.RowMapper;
import com.nt.bo.EmployeeBO;
public class ListEmpDaoImpl implements ListEmpDao {
    private static final String GET_EMP_DETAILS_BY_NO="select empno,ename,job,sal from
    emp where empno=?";
    private static final String GET_EMP_DETAILS_BY_DESG="select empno,ename,job,sal
from emp where job=?";
    private JdbcTemplate jt;
```

```
    public void setJt(JdbcTemplate jt) {
            this.jt = jt;
    }
    @Override
    public EmployeeBO getEmpDetailsByNo(int eno) {
      EmployeeBO empbo=jt.queryForObject(GET_EMP_DETAILS_BY_NO,
                                  1*                                  new Object[]{eno},
                                                                      new EmpRowMapper());
            return empbo;
    }
     private class EmpRowMapper implements RowMapper<EmployeeBO>{

            @Override
            public EmployeeBO mapRow(ResultSet rs, int pos) throws SQLException {
                System.out.println("mapRow(-,-)");
                    // Keep the record of ResultSet into BO class object
                    EmployeeBO ebo=new EmployeeBO(rs.getInt(1),
                                                                      rs.getString(2),
                                                                      rs.getString(3),
                                                                      rs.getInt(4));
                    return ebo;
            }//mapRow
     }//inner class
     @Override
    public List<EmployeeBO> getEmpDetailsByDesg(String desg) {

            List<EmployeeBO> list=jt.query(GET_EMP_DETAILS_BY_DESG,
                                                          new Object[]{desg},
            new ResultSetExtractor<List<EmployeeBO>>(){
    @Override
public List<EmployeeBO> extractData(ResultSet rs)
    throws SQLException,DataAccessException {
                    System.out
            .println("extract Data");
    List<EmployeeBO> list=new ArrayList<EmployeeBO>();
    while(rs.next()){
    EmployeeBO ebo=new EmployeeBO(rs.getInt(1),
                                  rs.getString(2),
                          rs.getString(3),
                                      rs.getInt(4));
                    list.add(ebo);
    }//while
return list;
}//extractData(-)
}//anymous inner class
            );//method call
            return list;
    }//method
}//class
```

How to write LAMBDA Expression based anonymous inner class?

```
EmployeeBO bo=null;
bo=jt.queryForObject(GET-EMP-DETAILS-BY-NO,
                     (ResultSet rs,int index)->{
                     ----
                     ----
},
no);
```

Lambda based inner class

note: we can use lambda based inner classes implementing interface only when that interface contain single method and it is not suitable if the interface contains more than one method

ListEmpService.java

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.EmployeeBO;
import com.nt.dao.ListEmpDao;
import com.nt.dto.EmployeeDTO;
public class ListEmpService {
    private ListEmpDao  dao;
    public void setDao(ListEmpDao dao) {
            this.dao = dao;
    }
            public  EmployeeDTO searchEmpByNo(int no){
            //use DAO class
            EmployeeBO ebo=dao.getEmpDetailsByNo(no);
            // Transpringframeworker BO data to DTO data
            EmployeeDTO dto=new EmployeeDTO(ebo.getNo(),
                                                    ebo.getEname(),
                                                    ebo.getJob(),
                                                ebo.getSal());
            if(dto!=null)
               return dto;
            else
                    return null;
    }
            public List<EmployeeDTO> searchEmpsByDesg(String desg){
             //use DAO class
            List<EmployeeBO> listbo=dao.getEmpDetailsByDesg(desg);
            // Copy List<BO> data to List<DTO>
            List<EmployeeDTO> listdto=new ArrayList<EmployeeDTO>();
            for(EmployeeBO bo:listbo){
                    EmployeeDTO dto=new EmployeeDTO(bo.getNo(),

bo.getEname(),
                                                    bo.getJob(),
                                                    bo.getSal());
                    listdto.add(dto);
            }//for
            if(listdto!=null)
                    return listdto;
            else
                    return null;
    }//method
}//class
```

what is difference between resultsetExtractor and RowCallbackHandler?
:-Both are usefull to customise multiple records given by select query execution but extractdata method of resultsetExtractor gives resultset object having all the retrieved records so we need to process them at once because it is stateless and does not remember query results across the multiple execution.
:- RowCallbackHandler is alternate to resultsetextractor having statefull behavior that means it remember retrieve records across the multiple execution by giving one record each time to the resultset object of processRow method.

EmployeeDTO.java

```
package com.nt.dto;
import java.io.Serializable;
public class EmployeeDTO  implements Serializable {
    private int no;
    private String name,job;
    private int sal;
    public EmployeeDTO() {
            // TODO Auto-generated constructor stub
    }
    public int getNo() {
            return no;
    }
public void setNo(int no) {
            this.no = no;
    }

    public String getName() {
            return name;
    }
    public void setName(String name) {
            this.name = name;
    }
    public String getJob() {
            return job;
    }
    public void setJob(String job) {
            this.job = job;
    }
    public int getSal() {
            return sal;
    }
    public void setSal(int sal) {
            this.sal = sal;
    }
    public EmployeeDTO(int no, String name, String job, int sal) {
            super();
            this.no = no;
            this.name = name;
            this.job = job;
            this.sal = sal;
    }
    @Override
    public String toString() {
            return "EmployeeDTO [no=" + no + ", name=" + name + ", job=" + job
                    + ", sal=" + sal + "]";
    }
    }
```

applicationContext.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.3.0.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="empDao"  class="com.nt.dao.ListEmpDaoImpl">
<property name="jt" ref="template"/>
</bean>
<bean id="empService"  class="com.nt.service.ListEmpService">
<property name="dao" ref="empDao"/>
</bean>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.ListEmpService;
public class ClientApp {
    public static void main(String[] args) {
      //create IOC Container
            ApplicationContext ctx=new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //get Bean
            ListEmpService service=(ListEmpService)ctx.getBean("empService");
            //call b.methods
            System.out.println("7499 Emp Details "+service.searchEmpByNo(7499));
            System.out.println("All CLERK Detials"+service.searchEmpsByDesg("CLERK"));
    }//main
}//class
```

Jarfiles:

|------>spjdbclib jarfile+ojdbc6.jar

Two methods of *JdbcTemplate* to work with *RowMapper, ResultSetExtrator* callback interface for the above requirement.

1) *public Object queryForObject (String sql, Object args[] RowMapper<T> rowmapper)*
(useful to get Bo class object having single record)
2) *public List<T> query(String sql,Object args[],ResultSetExtrator<T> extrator);*

In real time the enduser supplied data will be received by 'service' class in the form of *VO or DTO* class object data.After processing request *BO*class object will be prepared to pass to *DAO*. *DAO* uses *BO* class object data for persistence operations and gives another *BO* class object back to 'service' class. This service class converts *BO* class object data to *DTO* class object data to send to client.*VO, BO* class object are not 'serializable' to send over the network where as *DTO* class objects are serializable objects so we can send to network.

Batch processing/Updating

Instead of performing multiple insert, update, delete operations on Database table by sending same query to database software for multiple times it is recommended to take the support of batch processing. Which sends query to database software for 1 time and sets batch of values to query for one time to reduce network roundtrip between java application and database software.

While working with *JdbcTemplate* we will use *batchUpdate()* method for this.

1) batchUpdate(String sql, List<Object []>args)
2)batchUpdate(String sql,PreparedStatementBatchSetter psbs)
3)batchUpdate(String sql, List<object []> args, int [] args types) and etc.....

PreparedStatementBatchSetter is a callback interface that can be used to set value to prepared statement provided by *JdbcTemplate* for each of a number of updates in a batch using same sql. It allows programmer to set batch values of query parameters in customized form by gathering values from java objects like *BO* objects not from object[].

This callback interface gives 2 methods:

a)setValues(PreparedStatement ps,int index)

Allows to set values to query params by specifying its index and by specifying query index to execute.

b)**getBatchSize()**

Returns numeric value to specify this number of times that setValues(-,-) methods should execute.

eg: If*getBatchSize()* returns 5 then *setValues(-)* method executes for 5 times to assign 5 steps of values to batch. Later the query executes multiple times(5) in DBsoftware having those batch of values.

Note:

Batch processing is very useful in bulk registrations like batch of student registrations, bulk SMS delivery, Senses details insertion and etc...

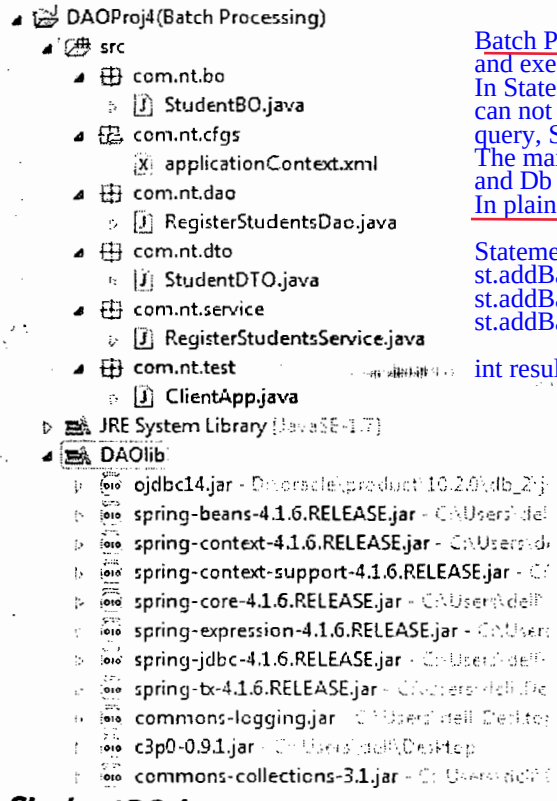
1) Batch processing/ Batch updation in SimpleStatement is adding multiple queries to the batch.

2) While working with PreparedStatement we can't add multiple queries to the batch because this always confines to single SQL query. But we can add multiple sets of query parameter values to the batch this reduce n/w roundtrips between java application and database software.

Sample code:

```
PreparedStatement ps = con.prepareStatement("insert into student values(?,?,?");
//set values to query params as batch
ps.setInt(1,00);
ps.setString(2,"raja");
ps.setString(3,"hyd");
ps.addBatch(); // add first set of values to batch

ps.setInt(1,01);
ps.setString(2,"ravi");
ps.setString(3,"vizag");
ps.addBatch(); // add secondset of values to batch
//execute
int result[]=ps.executeBatch();
```

In real time DAO gets batch values in the form of ListBO, setting these values to query params to query through batch is bit complex while working with plain jdbc to overcome this problem use BatchPreparedStetementSetter callback interface that simplifies the whole process.



Batch Processing/Updation: the process of keeping related queries/param values in the batch and executing them is called batch processing/updation.
In Statement object we can add multiple queries to the batch where as in PreparedStatement we can not multiple queries to batch becoz Prepared Statement obj is always confined single query, So we can multiple sets of query param values to the batch.
The main advantage of any batch processing reducing network round trips between java App and Db s/w.
In plain JDBC Simple Statement object:

```
Statement st=con.createStatement();
st.addBatch("query1");
st.addBatch("query2"); // must be non-select queries
st.addBatch("query3");

int result[]=st.executeBatch();
```

In Jdbc PreparedStatement object:

```
PreparedStatement obj ps=con.preparedStatement("insert into student values(?,?,?)");
ps.setInt(1,200);
ps.setString(2,"jani");
ps.setString(3,"hyd");
ps.addBatch();//adds 1st set values to Batch

ps.setInt(1,201);
ps.setString(2,"begum");
ps.setString(3,"hyd");
ps.addBatch();//adds 2nd set values to Batch

int result[]=ps.executeBatch();
```

usecase: the PreparedStatement based batch processing is usefull in group reservation of bus tickets. railways tickets and etc.
While working with JdbcTemplate we can use the folling methods for batch updation
a-:batchUpdate(String---sql)----:For simple Statement obj based batch processing
b:-int[] batchUpdate(Strng sql,List<Object[]>args)
c:-int[] batchUpdate(Strng sql,BatchPreparedStatementSetter bpss)| PreparedStatement
d:-int[] batchUpdate(Strng sql,List<Object[]>args,int[] argTypes) | based Batch

BatchPreparedStatementSetter (Callback interface to put multiples of sets of query param values to batch)
|---- getBatchSize();
|---- setValues(PreparedStatement ps,int index)

If getBatchsize() return 5 the set values method executes 5 number of types in each execution this method assign one set of values to query params in the batch where the index value increment in each exectution by 1.

StudentBO.java

```
package com.nt.bo;
public class StudentBO {
    private int sno;
    private String sname,sadd;
        public StudentBO() {
    }
public StudentBO(int sno, String sname, String sadd) {
    this.sno = sno;
    this.sname = sname;
```

```
        this.sadd = sadd;
    }

    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
    }
```

RegisterStudentsDao.java

```
package com.nt.dao;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util:List;
import org.springframework.jdbc.core.BatchPreparedStatementSetter;
import org.springframework.jdbc.core.JdbcTemplate;
import com.nt.bo.StudentBO;
public class RegisterStudentsDao {
    private static final String  STUD_INSERT_QRY="insert into student values(?,?,?)" ;
    private JdbcTemplate jt;
            public void setJt(JdbcTemplate jt) {
            this.jt = jt;
    }
    public boolean insert(final List<StudentBO> listbo){
            int result[]=jt.batchUpdate(STUD_INSERT_QRY, new BatchInsertion(listbo));
            if(result!=null)
                    return true;
            else
                    return false;
    }
            private static final class  BatchInsertion implements
BatchPreparedStatementSetter{
        private List<StudentBO> listbo;
            public BatchInsertion(List<StudentBO> listbo){
                    this.listbo=listbo;
```

```
        }
        @Override
        public int getBatchSize() {
                System.out.println("getBatchSize()");
                return listbo.size();
        }
        @Override
        public void setValues(PreparedStatement ps, int index)
                        throws SQLException {
                System.out.println("setValues(-,-)");
                ps.setInt(1,listbo.get(index).getSno());
                ps.setString(2,listbo.get(index).getSname());
                ps.setString(3,listbo.get(index).getSadd());
        }//setValues(-,-)
    }//class
}
```

RegisterStudentsService.java

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.StudentBO;
import com.nt.dao.RegisterStudentsDao;
import com.nt.dto.StudentDTO;

public class RegisterStudentsService {
    private RegisterStudentsDao dao;
    public void setDao(RegisterStudentsDao dao) {
        this.dao = dao;
    }
    public String  registerStudents(List<StudentDTO> listdto){
        //convert  ListDTO to ListBO object
        List<StudentBO> listbo=new ArrayList<StudentBO>();
        for(StudentDTO dto:listdto){
            StudentBO bo=new StudentBO(dto.getSno(),
                                       dto.getSname(),
                                       dto.getSadd());
            listbo.add(bo);
        }
        // Use DAO class
        boolean status=dao.insert(listbo);
        if(status){
            return "Batch insertion success";
        }
        else{
            return "Batch insertion failed";
        }
    }//method
}
```

StudentDTO.java

```
package com.nt.dto;
import java.io.Serializable;
public class StudentDTO implements Serializable {
    private int sno;
    private String sname,sadd;

    public StudentDTO() {
        }
    public StudentDTO(int sno, String sname, String sadd) {
        this.sno = sno;
        this.sname = sname;
        this.sadd = sadd;
    }

    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
}
```

applicationContext.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.3.0.xsd">
    <bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
    </bean>
    <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
    </bean>
```

```
<bean id="stDao"  class="com.nt.dao.RegisterStudentsDao">
<property name="jt" ref="template"/>
</bean>
<bean id="stService"  class="com.nt.service.RegisterStudentsService">
<property name="dao" ref="stDao"/>
</bean>
</beans>
```

Client.java

```
package test;
import java.util.ArrayList;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.dto.StudentDTO;
import com.nt.service.RegisterStudentsService;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC container
            ApplicationContext ctx=new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext
.xml");
     // get Bean
            RegisterStudentsService
service=ctx.getBean("stService",RegisterStudentsService.class);
    // Prpare Batch of Studnets
            StudentDTO st1=new StudentDTO(101,"raja","hyd");
            StudentDTO st2=new StudentDTO(102,"ravi","hyd");
            List<StudentDTO>  students=new ArrayList<StudentDTO>();
            students.add(st1); students.add(st2);
            //call B,method
            String result=service.registerStudents(students);
            System.out.println(result);
    }//main
}//class
```

Student table
|---->Sno (n) (pk)
|---->Sname (VC2)
|----->Sadd (VC2)

In order to use local variables or parameters of the method definition inside the inner class we must take them as final variables.

NamedParameterJdbcTemplate

Spring JDBC supports 2 types of params in the SQL Query

1) Positional parameter (?)
2) Named Parameter (:<name>)

JdbcTemplate supports only positional parameters(?). we need to use var-args values or Object[] values to assign values to positional parameters based on their indexes. In this process if we confuse towards identifying the indexes there is a possibility of getting problems.To overcome above problem the spring JDBC has given *NamedParameterJdbcTemplate* as wrapper around *JdbcTemplate* supporting Named parameters (:<varname>) in the query. This provides all the functionalities of *JdbcTemplate* but allows us to work with named parameters (:<varname>)

eg: select * from emp where empno>=:min and empno<=:max;

The underlying JDBC of spring JDBC converts each name parameter into positional parameter before executing the query.

There are 2 ways to set values to *NamedParameters*

1) As Map<String, Object> object

Specify the names of named parameters as keys and the objects as values in elements.

eg: Map<String, Object> map=new HashMap<String, Object>();

map.put("min", 10);

map.put("max", 100);

2) By using the implementation class objects of *SqlParameterSource(I)MapSqlParameterSource(I)*

Allows to specify the name, values of named parameters as the args of *addValue(-,-).*

SqlParameterSource params=new MapSqlParameterSource();

A:-

params.addValue("min", 10);

params.addValue("max", 100);

B:- BeanPropertySqlParameterSource

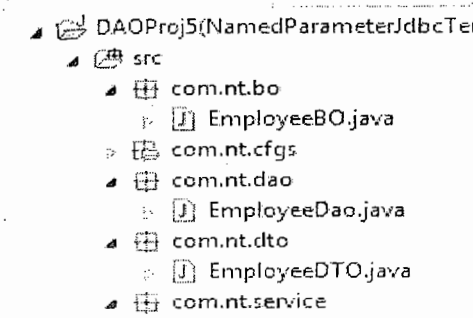
Note: *Allows to specify bean object values the named parameters values ~~and~~ But the names of named parameters and BeanProperty names(member variable) must match*

- NamedParameter JdbcTemplate internally uses JdbcTemplate to send and execute SQLQuery once it translate named parameters to the Jdbc style posistional parameters.
- To create NamedParameterJdbcTemplate class object we need DataSource object and it should be injected through constructor injection because there is no setDataSource method.

Example on BeanPropertySqlParameterSource:-

```
BeanPropertySqlParameterSource bpp=new BeanPropertySqlParameterSource();
Bean bean=new Bean();
bean.setMin(100);bean.setMax(200);
public class Bean{
    private int min;
    private int max;
//setter & getter
---
--
}
```

since database software understands only positinal parameter the NamedParameterJdbcTemplet translates given namedparameter into positional parameters and gives to wrapped JdbcTemplate to execute them in database software



EmployeeBO.java

```
package com.nt.bo;
public class EmployeeBO {
    private int eno,salary;
    private String ename,job;
    public EmployeeBO() {
    }
    public EmployeeBO(int eno, int salary, String ename, String job) {
            this.eno = eno;
            this.salary = salary;
            this.ename = ename;
            this.job = job;
    }
    public int getEno() {
            return eno;
    }
    public void setEno(int eno) {
            this.eno = eno;
    }
    public int getSalary() {
            return salary;
    }
    public void setSalary(int salary) {
            this.salary = salary;
    }
    public String getEname() {
            return ename;
    }
    public void setEname(String ename) {
            this.ename = ename;
    }
    public String getJob() {
            return job;
    }
    public void setJob(String job) {
            this.job = job;
    }
```

```
}
```

EmployeeDAO.java

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import com.nt.bo.EmployeeBO;
public class EmployeeDao {
    private static final String GET_EMP_NAME_QRY="select ename from emp where
    empno=:eno";
    private static final String GET_EMP_DETAILS_BY_NO_QRY="select empno,ename,job,sal
    from emp where empno=:eno";
    private static final String  EMP_INSERT_QRY="insert into emp(empno,ename,job,sal)
    values(:eno,:ename,:job,:salary)";
    private NamedParameterJdbcTemplate njt;
    public void setNjt(NamedParameterJdbcTemplate njt) {
            this.njt = njt;
    }
    public String findName(int no){
            //prepare param values
            Map <String,Object> map=new HashMap<String,Object>();
            map.put("eno",no);

            String name=njt.queryForObject(GET_EMP_NAME_QRY,
                                                        map,
                                                        String.class);
            return name;
    }
    public EmployeeBO  findEmpDetails(int no){
             //parpare param values
            MapSqlParameterSource params=new MapSqlParameterSource();
            params.addValue("eno",no);
            //execute Qurey
            EmployeeBO ebo=njt.queryForObject(GET_EMP_DETAILS_BY_NO_QRY,
                                                        params,
                                                        new
EmpRowMapper());
            return ebo;
    }
    private static final class EmpRowMapper implements RowMapper<EmployeeBO>{
            @Override
            public EmployeeBO mapRow(ResultSet rs, int pos) throws SQLException {
                    // Store ResultSet record into BO object
                     EmployeeBO ebo=new EmployeeBO(rs.getInt(1),
```

```
                                                                    rs.getInt(4),
                                                                    rs.getString(2),

rs.getString(3));
                    return ebo;
            }//mapRow
        }//inner class

        public  int  insertRecord(EmployeeBO ebo){
            //prepare param values
            BeanPropertySqlParameterSource params=
                            new BeanPropertySqlParameterSource(ebo);
            int result=njt.update(EMP_INSERT_QRY, params);
            return result;
        }
    }
```

EmployeeService.java

```
package com.nt.service;
import com.nt.bo.EmployeeBO;
import com.nt.dao.EmployeeDao;
import com.nt.dto.EmployeeDTO;
public class EmployeeService {
    private EmployeeDao dao;
    public void setDao(EmployeeDao dao) {
            this.dao = dao;
    }
    public String searchNameByNo(int no){
             //use DAO
            String name=dao.findName(no);
            return name;
    }
    public EmployeeDTO searchEmpByNo(int no){
            //use DAO
            EmployeeBO bo=dao.findEmpDetails(no);
            // convert BO into DTO
            EmployeeDTO dto=new EmployeeDTO(bo.getEno(),
                                    bo.getSalary();
                                                                    bo.getEname(),
                                                                    bo.getJob());
            return dto;
    }
    public String registerEmp(int no,String name,String job,int sal){
            //create DTO
             EmployeeDTO dto=new EmployeeDTO(no,sal,name,job);
            // convert DTO to BO
            EmployeeBO bo=new EmployeeBO(dto.getEno(),
                                                dto.getSalary(),
                                                dto.getEname(),
                                                dto.getJob());
```

```
        //use DAO
        int result=dao.insertRecord(bo);
        if(result==1)
                return "Regisration Success";
        else
                return "Registraion failed";
    }
}
```

EmployeeDTO.java

```
package com.nt.dto;
import java.io.Serializable;
public class EmployeeDTO implements Serializable {
    private int eno,salary;
    private String ename,job;
    public EmployeeDTO() {
    }
    public EmployeeDTO(int eno, int salary, String ename, String job) {
        this.eno = eno;
        this.salary = salary;
        this.ename = ename;
        this.job = job;
    }
    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }
    public String getJob() {
        return job;
    }
    public void setJob(String job) {
        this.job = job;
    }
    @Override
    public String toString() {
```

```
            return "EmployeeDTO [eno=" + eno + ", salary=" + salary + ", ename="
                    + ename + ", job=" + job + "]";
    }
}
```

applicationContext.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.3.0.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="ntemplate"
    class="org.springframework.jdbc.core.namedparam.NamedParameterJdbc
Template">
<constructor-arg  ref="dataSource"/>
</bean>
<bean id="empDao"  class="com.nt.dao.EmployeeDao">
<property name="njt" ref="ntemplate"/>
</bean>
<bean id="empService"  class="com.nt.service.EmployeeService">
<property name="dao" ref="empDao"/>
</bean>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.EmployeeService;
public class ClientApp {
    public static void main(String[] args) {
            //create IOC contianer
            ApplicationContext ctx =new
    FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //get Bean
            EmployeeService service=ctx.getBean("empService",EmployeeService.class);
            //call b.methods
            System.out.println("7499 emp name"+service.searchNameByNo(7499));
            System.out.println("7499 emp details"+service.searchEmpByNo(7499));
            System.out.println("Emp
    Registred?"+service.registerEmp(9999,"raja","clerk",8999));
    }
}
```

Note: *Spring2.5 is compatible with JDK ~~1.6~~ 1.4 but JdbcTemplate class is designed without having support of generics, var args, AutoBoxing features to overcome this problem SimpleJdbcTemplate class is given as a wrapper around JdbcTemplate supporting JDK 1.6 features. In spring 3 JdbcTemplate class is upgrated to JDK 1.6 features so SimpleJdbcTemplate class is* ***deprecated in spring 3.*** 1
(SimpleJdbcTemplate removed in spring 4.x)

Working with SimplejdbcInsert, SimpleJdbcCall

Uses metadata concepts as simplifies envinorment to call PL/SQL procedures and to insert records.

SimpleJdbcInsert

It uses Datebase table metadata concepts are used to insert records dynamically without insert SQL query. It just take table name, column name, column values from the programmers and uses them to generate dynamic SQL insert query to insert records. We can give these column names and values either as *Map* object or as *SQLParameterSource(I)* implementation class objects. The *Map* object contains column names as keys and column values as values.To create *SimpleJdbcInsert* object *DataSource* object is dependent object and we can *setTableName(-)* to specify dbtable name and *execute(-)* to specify column names, column values and also to execute the dynamically generated Insert SQL query.If our application deals with only registrations (insert operation) then we need to work with *SimpleJdbcInsert* class instead of *JdbcTemplate*.

- DAOProj6(SimpleJdbcInsert)
  - src
    - com.nt.bo
      - StudentBO.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.dao
      - StudentDao.java
    - com.nt.service
      - StudentService.java
    - com.nt.test
      - ClientApp.java
    - com.nt.vo
      - StudentVO.java
  - JRE System Library [JavaSE-1.7]
  - DAOlib

StudentBO.java

```
package com.nt.bo;
    public class StudentBO {
        private int sno;
        private String sname;
        private String sadd;
        public StudentBO() {
        }
        public StudentBO(int sno, String sname, String sadd) {
            this.sno = sno;
            this.sname = sname;
```

```
            this.sadd = sadd;
    }
    public int getSno() {
            return sno;
    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
            this.sadd = sadd;
    }
}
```

StudentDAO.java

```
package com.nt.dao;
import java.util.HashMap;
import java.util.Map;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
import com.nt.bo.StudentBO;
public class StudentDao {
    private SimpleJdbcInsert sji;
    public void setSji(SimpleJdbcInsert sji) {
            this.sji = sji;
    }
    public int  insert(StudentBO bo){
            //prepare params for execute(-)
             Map<String,Object> map=new HashMap<String,Object>();
             map.put("sno",bo.getSno());
             map.put("sname",bo.getSname());
             map.put("sadd",bo.getSadd());
             //specify table name
             sji.setTableName("student");
             //execute the logic
             int result=sji.execute(map);
             return result;
    }//insert(-)
}//class
```

```
@override
public int insert(StudentBo bo){
 int count=0;
BeanPropertySqlParameterSource bpsps=null;
//set table name
sji.setTableName("student");
//prepare BeanPropertySqlParameterSource class object
bpsps=new BeanPropertySqlParameterSource();
//execute query that is generated dynamycally
count=sji.execute(bpsps);
return count;
}
```

we can use this only when db table column names and bo class property names are matching

StudentService.java

```
package com.nt.service;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentDao;
import com.nt.vo.StudentVO;
public class StudentService {
    private StudentDao dao;
    public void setDao(StudentDao dao) {
            this.dao = dao;
    }
    public String registerStudent(String sno,String sname,String add){
            //prepare VO
            StudentVO vo=new StudentVO(sno,sname,add);
            //Convert VO object to BO object
            StudentBO bo=new StudentBO();
            bo.setSno(Integer.parseInt(vo.getNo()));
            bo.setSname(vo.getSname());
            bo.setSadd(vo.getSadd());
            //use DAO
            int result=dao.insert(bo);
            if(result==0)
                    return "Registration failed";
            else
                    return "Registraction success";
    }//method
}//class
```

StudentVO.java

```
package com.nt.vo;
public class StudentVO {
    private String no,sname,sadd;
            public StudentVO(String no, String sname, String sadd) {
            super();
            this.no = no;
            this.sname = sname;
            this.sadd = sadd;
    }
    public StudentVO() {
    }
    public String getNo() {
            return no;
    }
    public void setNo(String no) {
            this.no = no;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
```

```
            this.sname = sname;
        }
        public String getSadd() {
            return sadd;
        }
        public void setSadd(String sadd) {
            this.sadd = sadd;
        }
    }
```

applicationContext.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.xsd">
    <bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
    </bean>
    <bean id="simpleJdbcInsert"
class="org.springframework.jdbc.core.simple.SimpleJdbcInsert">
    <constructor-arg  ref="dataSource"/>
    </bean>
    <bean id="stDao"  class="com.nt.dao.StudentDao">
    <property name="sji" ref="simpleJdbcInsert"/>
    </bean>
    <bean id="stService"  class="com.nt.service.StudentService">
    <property name="dao" ref="stDao"/>
    </bean>
    </beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.StudentService;
public class ClientApp {
        public static void main(String[] args) {
            //Create IOC container
            ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
            //get ServiceBean
            StudentService service=ctx.getBean("stService",StudentService.class);
            //call B.method
            System.out.println("Result="+service.registerStudent("1010", "Ramesh","hyd"));
        }
}
```

VO class is a java bean that doesn't implements *Serializable(I)* and it's object represents input values coming to the controller Application.DTO class is java bean that implements *Serializable(i)* and it's object represents input values or output values to from 1 resource to another resource over the network.Generally the VO class objects contains String properties holding all the zip values given by enduser like form data, DTO class contains all types of Properties and BO class contains all types of properties (But these properties types and Database table column types generally should match) *SimpleJdbcInsert* internally uses *JdbcTemplate* while completing the persistence operation.*JdbcTemplate, SimpleJdbcTemplate, NamedParameterJdbcTemplate* class objects are thread safe once configure i.e they allow one thread at a time to perform persistence operations which degrades the performance. To overcome this problem we can use *SimpleJdbcInsert* or *SimpleJdbcCall* whose objects are multithreaded allowing multiple thread symultaniously.

Usecase:
When we ask visitors to register in timeframe (like online web counsiling) we need to allow symultanious record insertion for better performance so prefer using SimpleJdbcInsert in that situation.

SimpleJdbcCall

It is a multithreaded, reusable object representing call to PL/SQL procedure or function. It runs on metadata concepts and simplifies the code that is required to call procedure or function. We just need to provide procedure or function name with map of '*in*' parameters to call procedure or function and this returns a Map having '*out*' parameter names and values.Instead of writing common persistence logic in every module as sqlqueries, it is recommended to write only for one time in database software as PL/SQL procedure or function.
eg: Instead of writing authentication logic in every module separately, write it only once as PL/SQL procedure or function and use it in multiple modules of a project.

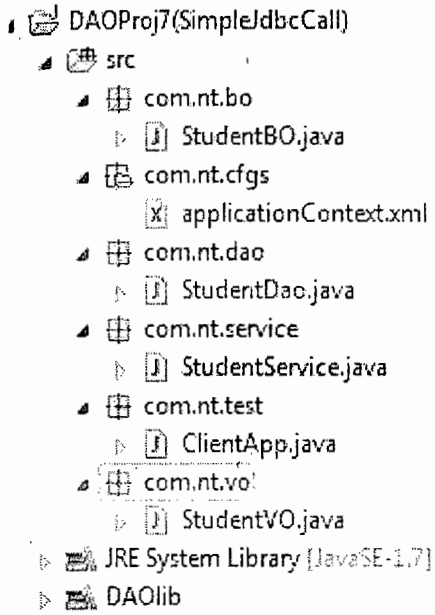
PL/SQL procedures in Oracle:

```
Create or replace procedure getDetails (no in number, name out varchar2, address out
varchar2) as
Begin
Select sname, sad into name, address from Student where sno=no;
end;
```

```
SimpleJdbcCall
|--->DS (dependent object--->constructor injection)
|--->execute(-)
    |--->Map as param (keys---> in param names
                       Values---> in param values)
    |--->Map as return value (keys---> out param names (comes as upper case letter)
                       Values---> out param values).
```

- In real time minimum complex PL/SQL procedure/functions must be developed by programmers and the complex PL/SQL procedure/functions must be developed by SQL developer.
- 50%--60% of persistence logic will be created in the form of PL/SQL procedure/function

for re-usability and for the centralization of the logic.



Asignment

Use SimpleJdbcCall support to call PL/SQL procedure or function that performs authentication.

StudentBO.java

```
package com.nt.bo;
public class StudentBO {
        private int sno;
        private String sname;
        private String sadd;
        public StudentBO() {
      }
      public StudentBO(int sno, String sname, String sadd) {
            this.sno = sno;
            this.sname = sname;
            this.sadd = sadd;
      }
      public int getSno() {
            return sno;
      }
      public void setSno(int sno) {
            this.sno = sno;
      }
      public String getSname() {
            return sname;
      }
      public void setSname(String sname) {
            this.sname = sname;
      }
      public String getSadd() {
            return sadd;
      }
      public void setSadd(String sadd) {
```

```
            this.sadd = sadd;
        }
}
```

StudentDAO.java

```
package com.nt.dao;
import java.util.HashMap;
import java.util.Map;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
import com.nt.bo.StudentBO;
public class StudentDao {
        private SimpleJdbcCall sjc;
        public void setSjc(SimpleJdbcCall sjc) {
            this.sjc = sjc;
        }
        public StudentBO getStudentDetails(int id){
            // set procedure name
            sjc.setProcedureName("getDetails");
            //Prepare IN params
            Map <String,Object> inParams=new HashMap<String,Object>();
            inParams.put("no",id);
            // call Pl/sql procedue
            Map<String,Object>outParams=sjc.execute(inParams);
            // Place OutParam values in BO object
            StudentBO bo=new StudentBO(id,
 (String)outParams.get("NAME"),
 (String)outParams.get("ADDRESS"));
            return bo;
        }//getStudentDetails(-)
}//class
```

*1 if in params name is matching with BO class property name then we can pass BO class object as the execute method argument

StudentService.java

```
package com.nt.service;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentDao;
import com.nt.dto.StudentDTO;
public class StudentService {
        private StudentDao dao;
        public void setDao(StudentDao dao) {
            this.dao = dao;
        }
        public StudentDTO seachStudentByNo(int no){
            //use DAO
            StudentBO  bo=dao.getStudentDetails(no);
            //Convert BO object to DTO object
            StudentDTO dto=new StudentDTO(bo.getSno(),
                bo.getSname(),
                bo.getSadd());
```

```
                return dto;
        }
}//class
```

StudentDTO.java

```
package com.nt.dto;
public class StudentDTO {
        private int no;
        private String sname,sadd;
                public StudentDTO(int no, String sname, String sadd) {
                super();
                this.no = no;
                this.sname = sname;
                this.sadd = sadd;
        }
public StudentDTO() {
        }
        public int getNo() {
                return no;
        }
        public void setNo(int no) {
                this.no = no;
        }
        public String getSname() {
                return sname;
        }
        public void setSname(String sname) {
                this.sname = sname;
        }
        public String getSadd() {
                return sadd;
        }
        public void setSadd(String sadd) {
                this.sadd = sadd;
        }
        @Override
        public String toString() {
                return "StudentDTO [no=" + no + ", sname=" + sname + ", sadd=" + sadd
                                + "]";
        }
}
```

applicationContext.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.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
```

```
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="simpleJdbcCall" class="org.springframework.jdbc.core.simple.SimpleJdbcCall">
<constructor-arg  ref="dataSource"/>
</bean>
<bean id="stDao"  class="com.nt.dao.StudentDao">
<property name="sjc" ref="simpleJdbcCall"/>
</bean>
<bean id="stService"  class="com.nt.service.StudentService">
<property name="dao" ref="stDao"/>
</bean>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.StudentService;
public class ClientApp {
        public static void main(String[] args) {
                //Create IOC container
                ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //get ServiceBean
                StudentService service=ctx.getBean("stService",StudentService.class);
                //call B.method
                System.out.println("1001 student details"+service.seachStudentByNo(1001));
        }//main
}//class
```

Mapping SQL operations as Sub Classes

This is technique in spring jdbc to perform persistence operations.This technique makes programmer to develop sub classes where we set *DataSource*, declare parameters and compile query only for 1 time but we execute that query multiple times by reusing the object of subclass for multiple times. This concept is similar to JDO (Java Data Objects) programming.Spring JDBC supplies *SqlQuery*, *SqlUpdate* classes to develop these sub classes. Use *SqlQuery* for select operations, use *SqlUpdate* for non-select operations.

Reason to go for subclass:

While working with *JdbcTemplate,NamedParameterJdbcTemplate, SimpleJdbcTemplate* classes if same query is executed for multipletimes then it creates multiple *JdbcpreparedStatements objects* internally on 1 *PrepareStatement* object for each execution basis. This is bad practice and also degrdes the performance. To overcome this problem we can use "Mapping SQL operations as sub classes" Technique.

SQLQuery:

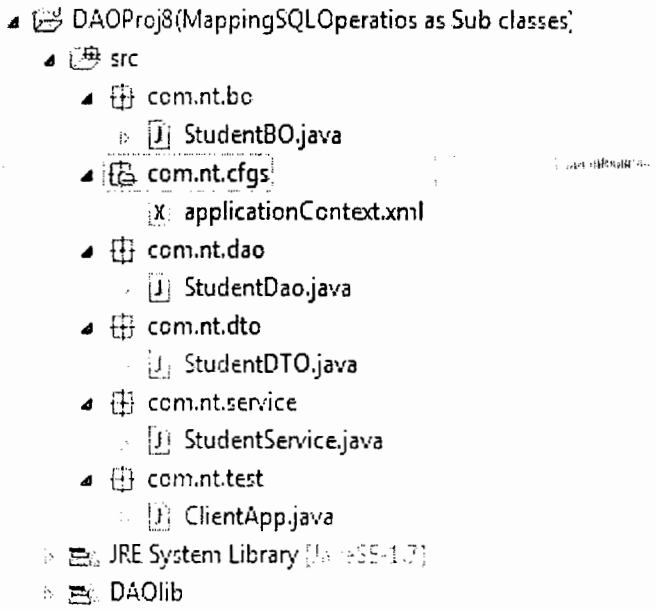
We develop separate subclass extending from *SqlQuery* for each select query inside DAO class.This subclass supplies *DataSource*, query to superclass to execute query for multiple

times by compiling the query only for 1 time. *SqlQuery* supplies execute() methods to executequery that gives list of objects (multiple records) and also give *finder(-)* methods to execute query that gives only one object (one record). This class also gives *mapRow(-,-)* method which can be used to process each record of *ResultSet* object in to another object. The *execute(),finder()* method internally it calls *mapRow(-,-)* method. Since *SqlQuery* in abstract class having more abstract methods we prefer using "*MappingSqlQuery*" which is subclass in the inheritance hierarchy of *SqlQuery*.While developing subclass for *SqlQuery/MappingSqlQuery* classes we use following methods in the subclasses.

a) declareParameter (-)
b) compile()
c) executeXxx(-)
d) findXxx(-) methods and etc.....

This technique say's give every thing to super class from subclass for query compilation and reuse subclass object. For multiple times to execute that query for multiple times.

- In this technique we need to take one separate inner class forever SQL query inside the inner class either extending from SqlQuery (or) SqlUpdate class. For example if DAO class want to execute 4 SQL queries we need to take 4 sub classes as inner classes of DAO class and each object of sub class represents 1 SQL query that is compiled. We can use that sub class object for multiple times in DAO class to execute the query for multiple times without doing compilation for multiple times.



StudentBO.java

```
package com.nt.bo;
public class StudentBO {
        private int sno;
        private String sname;
        private String sadd;
        public StudentBO() {
                }

        public StudentBO(int sno, String sname, String sadd) {
                this.sno = sno;
                this.sname = sname;
                this.sadd = sadd;
        }
        public int getSno() {
                return sno;
        }
        public void setSno(int sno) {
                this.sno = sno;
        }
        public String getSname() {
                return sname;
        }
        public void setSname(String sname) {
                this.sname = sname;
        }
        public String getSadd() {
                return sadd;
        }
        public void setSadd(String sadd) {
                this.sadd = sadd;
        }}
```

StudentDAO.java

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.SqlParameter;
import org.springframework.jdbc.object.MappingSqlQuery;
import org.springframework.jdbc.object.SqlUpdate;
import com.nt.bo.StudentBO;
public class StudentDao {
        private static final String GET_STUDETNS_BY_ADDRESS="select * from student  where
sadd=?";
        private DataSource ds;
        StudentSelector selector;
        StudentUpdator  updator;
```

```
        public StudentDao(DataSource ds) {
                this.ds=ds;
                selector=new StudentSelector(ds,GET_STUDETNS_BY_ADDRESS);
        }
        //DAO class method
        public List<StudentBO> findStudentsByAdderess(String addrs){
                List<StudentBO> listbo=selector.findByAddress(addrs);
                return listbo;
        }
private static final class StudentSelector extends MappingSqlQuery<StudentBO>{
                public StudentSelector(DataSource ds,String qry){
                        //supply dataSoruce,query details to super class
                        super(ds,qry);
                        super.declareParameter(new SqlParameter(Types.VARCHAR));
                        super.compile();
                }
                @Override
                protected StudentBO mapRow(ResultSet rs, int index) throws SQLException {
                        System.out.println("mapRow(-)");
                        return new StudentBO(rs.getInt(1),
                                                        rs.getString(2),
                                                        rs.getString(3));
                        }
                public List<StudentBO>  findByAddress(String addrs){
                        List<StudentBO> listbo=super.execute(addrs);
                        return listbo;
                }
        }//inner class/sub class
}//class
```

StudentService.java

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentDao;
import com.nt.dto.StudentDTO;
public class StudentService {
        private StudentDao dao;
        public void setDao(StudentDao dao) {
                this.dao = dao;
        }
        public List<StudentDTO> searchStudentsByAddress(String addrs){
                //use DAO
                List<StudentBO> listbo=dao.findStudentsByAdderess(addrs);
                //convert listBO object to listDTO object
                List<StudentDTO> listdto=new ArrayList<StudentDTO>();
                for(StudentBO  bo:listbo){
                        StudentDTO dto=new StudentDTO(bo.getSno(),
                                                                bo.getSname(),
```

```
                                                                bo.getSadd());
                        listdto.add(dto);
                }
                return listdto;
        } //method
        }//class
```

StudentDTO.java

```
package com.nt.dto;
public class StudentDTO {
        private int no;
        private String sname,sadd;
                public StudentDTO(int no, String sname, String sadd) {
                super();
                this.no = no;
                this.sname = sname;
                this.sadd = sadd;
        }
public StudentDTO() {
        }
        public int getNo() {
                return no;
        }
        public void setNo(int no) {
                this.no = no;
        }
        public String getSname() {
                return sname;
        }
        public void setSname(String sname) {
                this.sname = sname;
        }
        public String getSadd() {
                return sadd;
        }
        public void setSadd(String sadd) {
                this.sadd = sadd;
        }
    @Override
        public String toString() {
                return "StudentDTO [no=" + no + ", sname=" + sname + ", sadd=" + sadd
                                + "]";
        }
}
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.StudentService;
public class ClientApp {
```

```
        public static void main(String[] args) {
                //Create IOC container
                ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //get ServiceBean
                StudentService service=ctx.getBean("stService",StudentService.class);
                //call B.method
                System.out.println("Hyd student
details"+service.searchStudentsByAddress("hyd"));
}//main
}//class
```

applicationContext.xml

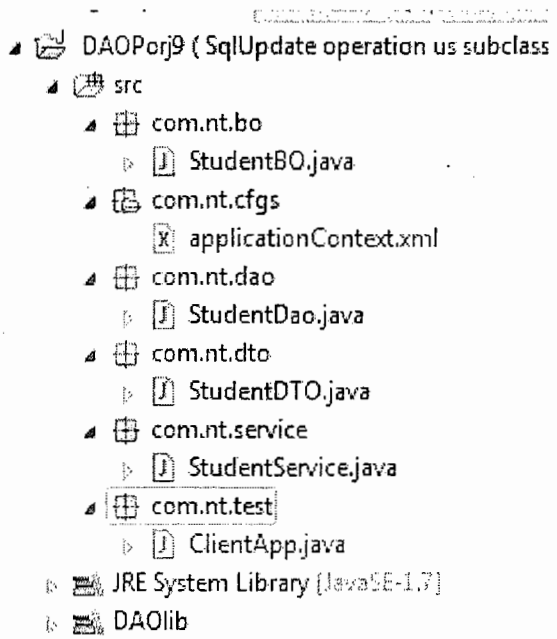
```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="stDao"  class="com.nt.dao.StudentDao">
<constructor-arg ref="dataSource"/>
</bean>
<bean id="stService"  class="com.nt.service.StudentService">
<property name="dao" ref="stDao"/>
</bean>
</beans>
```

SqlUpdate:

It is similar to *SqlQuery* class but given for non-select queries (update queries) execution. We can prepare subclass to *SqlUpdate* class inside the DAO class where we supply dataSource, query to *SqlUpdate* class only once. But we use these subclass object for multiple times to execute same nonselect query with the support of *update(-,-)* methods.*SqlUpdate* is a concrete class and supplies lots of *update(-)* methods to execute non-select query with same or different parameters for multiple times.For every non-select sqlquery we need to take separate sub class for *SqlUpdate* inside the DAO class.

Why should we learn SimpleJdbcInsert, SimpleJdbcCall and MappingSQL operations as subclasses techniques even though JdbcTemplate class is already available?

Ans: JdbcTemplate class object is single threaded object/ threadsafe object i.e it allows one thread at a time to perform the persistence operations. To overcome this problem use *SimpleJdbcInsert, SimpleJdbcCall* classes(which are multi threaded).When we use *JdbcTemplate* class object to execute same query for multiple times it internally creates multiple *PreparedStatement* object on one object for each execution basis which is a bad practice. To overcome this problem use MappingSql Operations as Subclasses technique.



StudentBO.java

```
package com.nt.bo;
public class StudentBO {
        private int sno;
        private String sname;
        private String sadd;
                public StudentBO() {
      }
  public StudentBO(int sno, String sname, String sadd) {
                this.sno = sno;
                this.sname = sname;
                this.sadd = sadd;
      }
      public int getSno() {
                return sno;
      }
      public void setSno(int sno) {
                this.sno = sno;
      }
      public String getSname() {
                return sname;
      }
      public void setSname(String sname) {
                this.sname = sname;
      }
      public String getSadd() {
                return sadd;
      }
      public void setSadd(String sadd) {
                this.sadd = sadd;
      }
}
```

StudentDAO.java

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.SqlParameter;
import org.springframework.jdbc.object.MappingSqlQuery;
import org.springframework.jdbc.object.SqlUpdate;
import com.nt.bo.StudentBO;
public class StudentDao {
	private static final String UPDATE_STUDENT_ADDRS_BY_NO="update student set
sadd=? where sno=?";
	private DataSource ds;
	StudentUpdator updator;
	public StudentDao(DataSource ds) {
		this.ds=ds;
		updator=new StudentUpdator(ds,UPDATE_STUDENT_ADDRS_BY_NO);
	}
		public int updateStudentDetails(int no,String newAddrs){
		int cnt=updator.updateDetails(no, newAddrs);
		return cnt;
	}
	private static final class StudentUpdator extends SqlUpdate{
				public StudentUpdator(DataSource ds,String qry) {
			//provide DataSoruce, Query to super class
			super(ds,qry);
			//Declare Parameters and given to super class
			super.declareParameter(new SqlParameter(Types.VARCHAR));
			super.declareParameter(new SqlParameter(Types.INTEGER));
			//compile the Query
			super.compile();
		}
		public int updateDetails(int no,String newAddrs){
			int cnt=super.update(newAddrs,no);
			return cnt;
		}
	}
}//class
```

StudentService.java

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentDao;
import com.nt.dto.StudentDTO;
```

```
public class StudentService {
	private StudentDao dao;
	public void setDao(StudentDao dao) {
		this.dao = dao;
	}
	public  String modifyStudentDetails(int no,String newAddrs){
		int cnt=dao.updateStudentDetails(no, newAddrs);
		if(cnt==0)
			return no+"  Student details are not updated";
		else
			return no+" student details are updated";
	}//method
}//class
```

StudentDTO.java

```
package com.nt.dto;
public class StudentDTO {
	private int no;
	private String sname,sadd;
		public StudentDTO(int no, String sname, String sadd) {
		super();
		this.no = no;
		this.sname = sname;
		this.sadd = sadd;
	}
	public StudentDTO() {
	}
	public int getNo() {
		return no;
	}
	public void setNo(int no) {
		this.no = no;
	}
	public String getSname() {
		return sname;
	}

	public void setSname(String sname) {
		this.sname = sname;
	}
	public String getSadd() {
		return sadd;
	}
	public void setSadd(String sadd) {
		this.sadd = sadd;
	}
	@Override
	public String toString() {
		return "StudentDTO [no=" + no + ", sname=" + sname + ", sadd=" + sadd
```

```
                                + "]";
        }
}
```

applicationContext.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.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="stDao"  class="com.nt.dao.StudentDao">
<constructor-arg ref="dataSource"/>
</bean>
<bean id="stService"  class="com.nt.service.StudentService">
<property name="dao" ref="stDao"/>
</bean>
</beans>
```

Client.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.StudentService;
public class ClientApp {
        public static void main(String[] args) {
                //Create IOC container
ApplicationContext ctx=new FileSystemXmlApplicationContext
                ("src/com/nt/cfgs/applicationContext.xml");
                //get ServiceBean
StudentService service=ctx.getBean("stService",StudentService.class);
                //call B.method
                System.out.println(service.modifyStudentDetails(1001, "hyd11"));
                }//main
}//class
```

AOP (Aspect Oriented Programming)

AOP is the methodology of programming that makes the programmer primary logics of application from the secondary logics or helper logics.

- ⇨ AOP is not replacement for OOP more ever it complements OOP
- ⇨ In OOP we develop business methods in classes by mixing up both primary, secondary logics. Due to this business methods becomes heavy and the reusability of secondary logics will be killed and we cannot enable or disable secondary logics without touching source code of business method.
- ⇨ To overcome above method use AOP that makes the programmer to write primary logics in separate class and secondary logics in separate class and allows to mix and bind both logics dynamically at runtime by generating new classes.

OOP example:

```
class Bank{
public boolean withdraw (int accntno, int amt)
{
//security logic
//logging logic
//Transaction logic
bal=bal-amt;    //primary logic
}
public boolean deposite(int accntno, int amt)
{
//security logic
//logging logic
//Transaction logic
bal=bal+amt;    //primary logic
}
}
```

Problems with oop:

1) kills the reusability of secondary logics

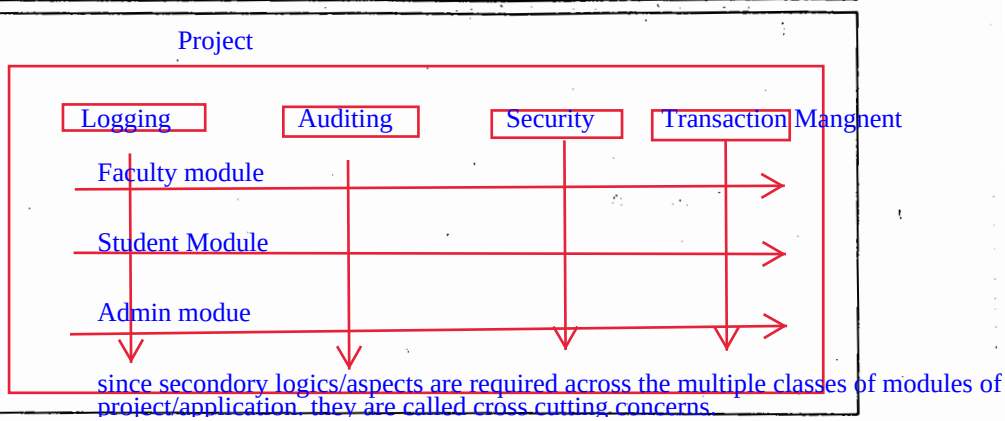
2) secondary logics dominate the primary logics

3) enabling or disabling secondary logics is quite complex because we need to modify source code of the project

AOP example:

```
class Bank
{
 withdraw (accno, amt)
{
bal=bal-amt;  // primary logic
}
deposit (actno,amt)
{
bal=bal+amt; // primary logic
}
}
class Security
```

```
{
.............
.............
...........
}
class Logging
{
.............
..............
}
```




```
class Bank{
 withdraw(acno, amt){
  bal=bal-amt; //primary logic
 }
 deposit(acno, amt){
  bal=bal+amt; //primay logic
 }
}
```

```
class Security{
 .... security
 .... Logic
}  (Secondary
      Logic)
```

```
class Logging{
 ...  (Logging
 ...   Logic)
}
```

AOP Implementor S/w

XML/Annotation cfgs

```
Prxoy class
class Bank$Pxoxy{

withdraw(acno,amt)
{
 Securiyt logic
 Logging Logic
 bal=bal-amt;
}
deposite(accno,amt){
 Security Logic
 Logging Logic
 bal=bal+amt;
}
}
```

Secondary logis also called as aspect cross cutting concern or middleware services

AOP advantages:

1) Secondary logics are not mixedup with primary logics.

2) Secondary logics become reusable logics.

3) By giving instructions to AOP enabled software we can enable (or) disable secondary logics.

4) We can add secondary logics dynamically at runtime without touching the source code of primary logics (using XML/@Annotation)

- The logic that is minimum to create the task is called the primary logic.Without primary logic task will not be completed.
- The logic that supports primary logic and makes primary logic as better logic but optional to have in the application is called secondary logic.
- While performing withdraw amount operation deducting amount from balance is called primary logic.
- Checking the identity of user, logging, auditing and etc are called secondary logics.Since we need secondary logic in multiple component classes of application or project they can be also called as cross cutting concerns because they need to be applied across the multiple classes.
- For example scenario that speaks about need of AOP with bank account use case.

- AOP is not replacement for OOP, moreever it complements OOP.The key component in OOP is class. The key component in AOP Aspect (cross cutting that represents primary logics).
- The language that is designed based on OOP principle like abstraction, encapsulation, inheritance and etc... is called OOP enable languages.
- Similarly OOP principle we also have AOP principles. The software that supports and implements AOP principles is called AOP enable software. AOP is not just specific to spring.
- There are multiple AOP enabled softwares like spring AOP, AspectJAOP,JBOSS AOP, JAC (java Aspect component), mugzilla AOP and etc...

AOP principles/Terminologies:

1. Aspect
2. JoinPoint
3. Advice
4. Pointcut
5. Target class
6. Weaving/Aspect Weaving
7. Proxy class

Aspect:

The logic that has to be applied across the multiple classes of appliaction is called Aspect. This logics is also called as crosscutting concern, secondary logic, middleware service logic.
eg: Security, logging, transaction and etc...

JoinPoint:

The possible points in classes on which aspect logics can be applied is called joinpoints. They are like fields, constructors, methods and etc....
Spring supports only methods as joinPoint.

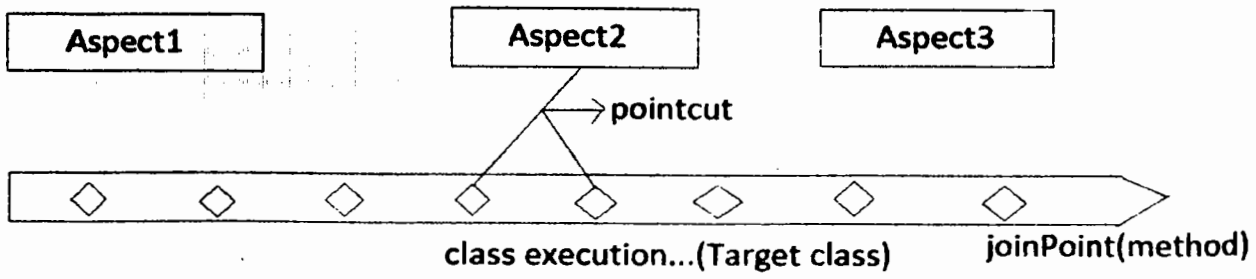
Advice:

It specifies or indicates how an aspect should be applied on JoinPoint like applying aspect before executing method or appling at the end of method and etc.....

Note:

Advice indicates that action tacken by aspect and join points.
There are 4 types of advices like:-

1) BeforeAdvice
2) AfterAdvice
3) ThrowsAdvice,
4) AroundAdvice.



PointCut:
It is a collection of JoinPoint on which aspects are adviced (i.e aspects are configured).

Target Class:
The class on which we want to advice the aspects. It is pre-AOP java class having primary logics in methods. If we call methods of target class only primary logic will be executed.

Weaving:
The process of advising aspects on the target class to build proxy class is called weaving. It is all about combining multiple things to build one 1 proxy class.

Proxyclass:
- The outcome class of weaving process where primary logics and secondary logics of aspects will be mixed up dynamically at runtime. It is post-AOP class. If we call methods on proxy class object the primary logic will be executed along with secondary logic
- Aspect weaver is an unit of AOP forward like spring AOP, AspectAOP and etc.. It uses some runtime libraries like jdk libraries or cglib libraries to generate proxy classes through weaving.
- This proxy classes are dynamically generated inmemory classes.

Spring1.x gives spring AOP to work with AOP.From spring 2.x AspectAOP has been integrated with spring. So spring 2.x we can use spring AOP and AspectJAOP to work with AOP concepts.

There are 4 approaches to implement AOP concepts in spring envinorment.
- Spring AOP Declaraive (xml)
- Spring AOP programmatic (javacode)
- Spring integrated AspectJ AOP declarative (xml)
- Spring integrated AspectJ AOP Annotatins (@)

There are 4 types of advices in AOP style programming.

1) AroundAdvice:
Here advice logic executes Around the target class method, i.e before and after target class method (Target method)

2) BeforeAdvice:
Here Advice logic executes before executing the target class method. So once target method is executed control does not come back to advice logic/method.

3) AfterAdvice:
Here advice logic executes after executing target method and before it returns the control back to caller.

4) ThrowsAdvice:
Here the Advice logic executes when exception is raised in target method.

Spring AOP declarative Approach:
Here we use xml file to configure Aspect, target classes and to configure the classes that perform weaving.

Around Advice:

Advice indicates the action taken by Aspect on JoinPoint. So Around advice makes its aspects to execute around the JoinPoint i.e Aspect/Advice logic executes before entering to target method and after executing the target method.

- This Around advice is useful to implement "perfomance" monitoring, caching, logging and etc usecase.

To develop this Advice/Aspect we need to take a class that implements MethodInterceptor(i) should provoke / invoke (methodInvocation invocat) as shown below.

Sample Code:

```
    public class MyAdvice implements MethodInterceptor{
    public Object invoke (MethodInvocation invocation)
    {
    ..............................................
    ........................ //logic that executes before entering into target method.
    Object retval=invocation.proceed(); //calls target method
    ..............................
    ..............................
    //logic that executes after finishing the execution cf target method.
    ......................................
    ................
    }
}
```

Using MethodInvocation object we can call getName(), getArguments() method names and it's arguments. We can call proceed() method to call target method from advice class.

3 important ponts/control points of around advice

=>Can access and modify target method arguments.

=>Can control target method execution i.e we can skip or proceed target method execution being from advice class.

=>Can access and modify the target method return value.

Jar file: IOCLib jar file+ spring-aop-<ver>.Release.jar+spring-aspect-<ver>.Release.jar+com.springsource.org/aopallionce-1.0.0.jar

Procedure to spring AOP Application in Declarative Approach:

a) Develop target class having method with primary logic.
b) Develop Aspect/Advice class having secondary logic.
c) Perform configures in springbean configures file for weaving
   i) Configure target class
   ii) Aspect/Advice class
   iii) Configure proxyFactoryBean class supplying target, Aspect classes to generate proxy class through weaving
d) Develop clientApp for testing
   i) Create IOC container
   ii) Get proxy class object by supplying the bean id of "proxyFactoryBean"
   iii) Call target class methods on proxy object.

In spring JoinPoints are methods of classes where Aspects can be adviced.

Pointcut specifies which aspect or aspects should be applied on which JoinPoint or JoinPoints.

Flow of execution with respect to AOP proj1 by invoking the generated proxy class:

- AOPProj1 (Spring AOp-Decl-AroundAdvice)
  - src
    - com.nt.aspect
      - CacheAdvice.java
      - LogAroundAdvice.java
      - PerformanceMonitoringAdvice.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.service
      - IntrAmtCalculator.java
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - AOPlib
    - spring-aop-4.1.6.RELEASE.jar - C:\Users\de
    - spring-aspects-4.1.6.RELEASE.jar - C:\Users
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\
    - spring-context-4.1.6.RELEASE.jar - C:\Users
    - spring-context-support-4.1.6.RELEASE.jar -
    - spring-core-4.1.6.RELEASE.jar - C:\Users\de
    - spring-expression-4.1.6.RELEASE.jar - C:\Us
    - commons-logging.jar - C:\Users\dell\Desk
  - **com.springsource.org.aopalliance-1.0.0.jar**

the Methodinvocation objedtc holds target method details,Similarly in case of Before advice,After advice,throws advice the Method class obj and Object class obj[] holds target methods details and we can these details in advice class while developing secondary logics.

CacheAdvice.java (usecase1)

```
package com.nt.aspect;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class CacheAdvice implements MethodInterceptor {
        Map <String,Object> cache=new HashMap<String,Object>();
        @Override
        public Object invoke(MethodInvocation invocation) throws Throwable {
      String
key=invocation.getMethod().getName()+Arrays.toString(invocation.getArguments());

      Object retValue=null;
       if(!cache.containsKey(key)){
          retValue=invocation.proceed();
          cache.put(key, retValue);
          System.out.println("From target method");
          return retValue;
       }
       retValue=cache.get(key);
       System.out.println("From cache");
```

```
        return retValue;
        }
}
```

LogAroundAdvice.java (usecase2)

```
package com.nt.aspect;
import java.util.Arrays;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class LogAroundAdvice implements  MethodInterceptor{
        @Override
        public Object invoke(MethodInvocation invocation) throws Throwable {
          System.out.println("Entering into "+invocation.getMethod().getName()+" with
args"+Arrays.toString(invocation.getArguments()));
    // Modify argument values
          Object args[]=invocation.getArguments();
          if(((Float)args[0])<50000){
                  args[1]=1.0f;
          }
          Object retValue=null;
          if( ((Float)args[0])==0.0f || ((Float)args[1])==0.0f || ((Float)args[2])==0.0f){
                    return 0.0f;
            }
          else{
            retValue=invocation.proceed();
          }
          System.out.println("Leaving from "+invocation.getMethod().getName()+" with
args"+Arrays.toString(invocation.getArguments()));
          if(((Float)retValue)<=25000)
                    return retValue;
          else
                  return ((Float)retValue)+(((Float) retValue)*0.1f);
        }
        }
```

PerformanceMonitoringAdvice.java (usecae3)

```
package com.nt.aspect;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class PerformanceMonitoringAdvice implements MethodInterceptor{
        long start,end;
        @Override
        public Object invoke(MethodInvocation invocation) throws Throwable {
                start=System.currentTimeMillis();
                Object retValue=invocation.proceed();
                end=System.currentTimeMillis();
                System.out.println(invocation.getMethod().getName()+"has taken "+(end-
start)+"ms to complete execution");
                return retValue;
        }
}
```

IntrAmtCalculator.java

```
package com.nt.service;
public  class IntrAmtCalculator {
        public  float calcIntrAmt(float principle,float rate,float time){
                System.out.println("Target class : Method");
                return  (principle*rate*time)/100.0f;
        }
}
```

applicationContext.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-2.0.xsd">
<bean id="target" class="com.nt.service.IntrAmtCalculator"/>
<bean id="logAdvice"  class="com.nt.aspect.LogAroundAdvice"/>
<bean id="pmAdvice"  class="com.nt.aspect.PerformanceMonitoringAdvice"/>
<bean id="cacheAdvice" class="com.nt.aspect.CacheAdvice"/>
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target" ref="target"/>
<property name="interceptorNames">
<list>
<value>pmAdvice</value>
<value>logAdvice</value>
<value>cacheAdvice</value>
</list>
</property>
</bean></beans>
```

Client Application.java

```
package test;
import java.util.Arrays;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.IntrAmtCalculator;
public class ClientApp {
  public static void main(String[] args) {
        //create IOC container
        ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //get Bean
        IntrAmtCalculator proxy=ctx.getBean("pfb",IntrAmtCalculator.class);
        //Call B.method
                float intrAmt=proxy.calcIntrAmt(80000,2,20);
                System.out.println("IntrAmt ="+intrAmt);
                System.out.println("---------------------------");
               float intrAmt1=proxy.calcIntrAmt(80000,2,20);
                System.out.println("IntrAmt ="+intrAmt1);
               System.out.println("---------------------------");
```

```
                float intrAmt2=proxy.calcIntrAmt(90000,2,20);
                System.out.println("IntrAmt ="+intrAmt2);
              System.out.println("----------------------------");
  }//main
}//class
```

- While working with AOP our target class must not be final and target class methods, must not be static, final, because it gives problem towards proxy class by extending from target class and overriding target class methods in proxy class.
- It is always recommended to take separate class for every cross cutting (secondary) logic. It is better not to mixup multiple logics in a single class.
- Aspect Weaver uses either CGLIB libraries (or) JDK libraries to generate Proxy class as Runtime Inmemory class.

The best use cases for around advice are

i) Performance monitoring
ii) Caching
iii) Around logging.

- Cache/buffer is a temporary memory that holds data for temporary period. In client-server application the cache at clientside holds server Application results and uses it across the multiple same request to reduce network round trip between client and server.
- In AOP programming we store target method result in a cache and we use that result across the multiple same method calls with same arguments. We generally take either array or collections (HashMap) as cache.
- While configuring cache Aspect/Advice along with other Aspect/Advice it is recommended to configure then after cache advice because if target method is getting it's result from cache there is no need of performing logging, performance monitoring.

Proxy interface:

- The interface that implemented by target class is called ProxyInterface.It is actually called as proxy interface because the dynamically generated proxy class implements this Interface. We need to configure this interface while configure "ProxyFactoryBean" class in xml file.
- Once proxy interface is configure in xml file the jdk libraries will be used to generated the proxy class dynamically, at runtime and this generated proxy class doesn't extends from target class ratherly implements proxy interface. Due to this here we can take target class a final class and target methods as final methods.

Note:-when multiple around advice is applied on 1 target method they will be executed in the order of configuration while intering into the method and same thing will be executed in reverse order while exiting from method.

- AOPProj2-SpringAOPDecl-AroundAdvice-ProxyInterfaces
  - src/main/java
    - com.nt.aspect
      - CacheAdvice.java
      - LogAroundAdvice.java
      - PerformanceMonitoringAdvice.java
    - com.nt.beans
      - IntrAmtCalculator.java
      - IntrAmtCalculatorInterface.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.commons
      - log.properties
    - test
      - AopAroundAdviceTest.java
  - src/test/java
  - JRE System Library [J2SE-1.5]
  - Maven Dependencies
  - src
  - target
  - logAround.html
  - pom.xml

CacheAdvice.java

```
package com.nt.aspect;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class CacheAdvice implements MethodInterceptor {
  private Map<String,Float> cache=new HashMap<String,Float>();
      public Object invoke(MethodInvocation invocation) throws Throwable {
            String key=null;
            Object retVal=null;
            System.out.println(invocation.getClass());
            //prepare targe method with args as the key

      key=invocation.getMethod().getName()+Arrays.toString(invocation.getArguments());
            //check for the result in cache
            if(!cache.containsKey(key)){
                  System.out.println("from target method");
                  retVal=invocation.proceed();  //calls target method
                  cache.put(key,((Float)retVal));
            }
            else{
                  System.out.println("from cache");
```

```
                retVal=cache.get(key);
            }
            return retVal;
        }//method
}//class
```

LogAroundAdvice.java

```
package com.nt.aspect;

import java.util.Arrays;
import java.util.Date;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

publicclass LogAroundAdvice implements MethodInterceptor {
privatestatic Logger logger=Logger.getLogger(LogAroundAdvice.class);
public LogAroundAdvice() {
        PropertyConfigurator.configure("src/main/java/com/nt/cominons/log.properties");
        }
        public Object invoke(MethodInvocation invocation) throws Throwable {
            Object retVal=0;
            Object args[]=null;
            logger.warn("Entering into "+invocation.getMethod().getName()+
                                    " with
args"+Arrays.toString(invocation.getArguments())+
                                    "at "+new Date());
            //modify target method args
            args=invocation.getArguments();

            if(((Float)args[0])<10000)
                args[1]=(Float)args[1]-0.5f;

            if(((Float)args[0])<=0 || ((Float)args[1])<=0 ||((Float)args[2])<=0)
                thrownew IllegalArgumentException("inputs must be +ve");

            retVal=invocation.proceed();

            logger.warn("Exited from "+invocation.getMethod().getName()+
                " with args"+Arrays.toString(invocation.getArguments())+
                "at "+new Date());

            if(((Float)retVal)>=5000)
                    retVal=((Float)retVal)+ ((Float)retVal)*0.05f;

            return retVal;
        }

}
```

PerformanceMonitoringAdvice.java

```
package com.nt.aspect;

import java.util.Arrays;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

publicclass PerformanceMonitoringAdvice implements MethodInterceptor  {
privatelong start=0;
privatelong end=0;
	public Object invoke(MethodInvocation invocation) throws Throwable {
		//seconday logic
		Object retVal=null;
		start=System.currentTimeMillis();  //pre logic
		retVal=invocation.proceed();  //calls target method
		end=System.currentTimeMillis();  //post logic
		System.out.println(invocation.getMethod().getName()+"with args
"+Arrays.toString(invocation.getArguments())+" has taken "+(end-start)+" ms");
		return retVal;
	}

}
```

IntrAmtCalculator.java

```
package com.nt.beans;

publicfinalclass IntrAmtCalculator implements  IntrAmtCalculatorInterface {

	publicfinalfloat calcIntrAmt(float pAmt,float rate,float time){
		//primary logic
		return  (pAmt*rate*time)/100.0f;
	}//method
}//class
```

IntrAmtCalculatorInterface.java

```
package com.nt.beans;

publicinterface IntrAmtCalculatorInterface {
	publicfloat calcIntrAmt(float pAmt,float time,float rate);

}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<!-- Configure Target class -->
<bean id="iac"   class="com.nt.beans.IntrAmtCalculator"/>
```

```
<!-- Configure Aspect/Advice -->
<bean id="pmAdvice"  class="com.nt.aspect.PerformanceMonitoringAdvice"/>
<bean id="logAdvice"   class="com.nt.aspect.LogAroundAdvice"/>
<bean id="cacheAdvice"  class="com.nt.aspect.CacheAdvice"/>
<!-- Configure ProxyFactory Bean to get PRoxy object -->
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target"  ref="iac"/>
<property name="proxyInterfaces">
<list>
<value>com.nt.beans.IntrAmtCalculatorInterface</value>
</list>
</property>
<property name="interceptorNames">
<list>
<value>cacheAdvice</value>
<value>pmAdvice</value>
<value>logAdvice</value>
</list>
</property>
</bean>

</beans>
```

log.properties

```
# for FileAppender,HTMLLayout
log4j.rootLogger=WARN,S
log4j.appender.S=org.apache.log4j.FileAppender
log4j.appender.S.File=logAround.html
log4j.appender.S.Append=true
log4j.appender.S.layout=org.apache.log4j.HTMLLayout
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 07:53:46 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
```

```
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>282</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:53:46 IST 2016</td>
</tr>

<tr>
<td>284</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:53:46 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 07:54:19 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>
```

```
<tr>
<td>279</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:54:19 IST 2016</td>
</tr>

<tr>
<td>280</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:54:19 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 07:55:13 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>283</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:55:13 IST 2016</td>
```

```
</tr>
<tr>
<td>284</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:55:13 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 07:56:25 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>283</td>
<td title="main thread">main</td>
<td title="Level">&lt;font color="#993300"&gt;&lt;strong&gt;WARN&lt;/strong&gt;&lt;/font&gt;&lt;/td&gt;
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
07:56:25 IST 2016</td>
</tr>

<tr>
<td>284</td>
<td title="main thread">main</td>
<td title="Level">&lt;font color="#993300"&gt;&lt;strong&gt;WARN&lt;/strong&gt;&lt;/font&gt;&lt;/td&gt;
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
```

```
07:56:25 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:16:29 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>294</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:16:29 IST 2016</td>
</tr>

<tr>
<td>303</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:16:29 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
```

```
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:26:01 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>282</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:26:01 IST 2016</td>
</tr>

<tr>
<td>283</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:26:01 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
```

```
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:26:16 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>277</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:26:16 IST 2016</td>
</tr>

<tr>
<td>279</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:26:16 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:27:16 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
```

```
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>280</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:27:16 IST 2016</td>
</tr>

<tr>
<td>281</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:27:16 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:27:26 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
```

```
<td>272</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:27:26 IST 2016</td>
</tr>

<tr>
<td>273</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:27:26 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:30:31 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>283</td>
<td title="main thread">main</td>
<td title="Level">><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:30:31 IST 2016</td>
</tr>
```

```
<tr>
<td>284</td>
<td title="main thread">main</td>
<td title="Level"><font color="#993300"><strong>WARN</strong></font></td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:30:31 IST 2016</td>
</tr>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Log4J Log Messages</title>
<style type="text/css">
<!--
body,table {font-family: arial,sans-serif; font-size: x-small;}
th {background: #336699; color: #FFFFFF; text-align: left;}
-->
</style>
</head>
<body bgcolor="#FFFFFF" topmargin="6" leftmargin="6">
<hr size="1" noshade>
Log session start time Wed Sep 14 08:30:33 IST 2016<br>
<br>
<table cellspacing="0" cellpadding="4" border="1" bordercolor="#224466" width="100%">
<tr>
<th>Time</th>
<th>Thread</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
</tr>

<tr>
<td>255</td>
<td title="main thread">main</td>
<td title="Level">&lt;font color="#993300"&gt;&lt;strong&gt;WARN&lt;/strong&gt;&lt;/font&gt;</td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Entering into calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
08:30:33 IST 2016</td>
</tr>

<tr>
<td>256</td>
<td title="main thread">main</td>
<td title="Level">&lt;font color="#993300"&gt;&lt;strong&gt;WARN&lt;/strong&gt;&lt;/font&gt;</td>
<td title="com.nt.aspect.LogAroundAdvice category">com.nt.aspect.LogAroundAdvice</td>
<td title="Message">Exited from calcIntrAmt with args[20000.0, 2.0, 20.0]at Wed Sep 14
```

08:30:33 IST 2016</td>
</tr>

Note: When proxy interface is not configurerd the CGLib library will be used to generate proxy class extending from target class.

Before Advice:

- This advice executes before begining the execution of target method and one advice method is executed then control automatically goes to target method.
- After completing the execution of target method control doesn't comeback to before advice.

To develop beforeAdvice our class must implements MethodBeforeAdvice(i) and should provide implementation to before (-,-,-) method.

- The use cases of beforeAdvice are Auditing, Security check and etc...

Method Signature:

public void before(Method method,Object args[],Object target)
Method method :- represents targetmethod.
Object args[] :- represents target arguments.
Obect target:- represents target class object

The important /control point of beforeAdvice:

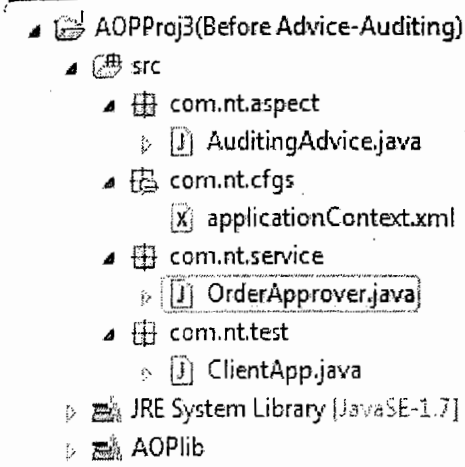
1) Can access and modify target method arguments.
2) Can not access and modify target method return value because the control doesn't come back to before advice once the target method execution is completed.
3) Can not control target method execution being from advice but can stop/abort target method execution by throwing exception in the before (-,-,-) method.

- Logging keep tracks of various components that are involved in the flow of execution of in Application.
- Auditing keeps track of various activities that are taken place in the execution of application.
- Generally the audit info will be written to the audit log file or audit table, it's like username,
  Ordertime, ordered and etc...

Note:

1) Keeping track of classes, methods, blocks that are involed in the Application execution is called logging.

2) Keeping track of various activities of end user and his operations in the application is called auditing.



+ com.SpringSource.org.aopalliance 1.0.0

AuditingAdvice.java

```
package com.nt.aspect;
import java.io.FileWriter;
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.Date;
import org.springframework.aop.MethodBeforeAdvice;
public class AuditingAdvice implements MethodBeforeAdvice {
        @Override
        public void before(Method method, Object[] args, Object target)
                        throws Throwable {
                System.out.println("Target class name"+target.getClass());
                System.out.println("Target method name"+method.getName());
                System.out.println("Target method args"+Arrays.toString(args));
                if((Integer)args[0]==0)
                        throw new IllegalArgumentException("Wrong Order id ");

                if((Integer) args[0]<0)
                         args[0]=((Integer)args[0])*-1;
                //Auditng
                  FileWriter fw=new FileWriter("E:\\Spring\\AuditLog.txt",true);
                  fw.write(args[0]+" order has come for approval at"+new Date()+"\n");
                  fw.flush();
                  fw.close();
 }
}
```

OrderApprover.java

```
package com.nt.service;
import java.util.Calendar;
public class OrderApprover {
                public String approveOrder(int orderid){
                // get system date
                Calendar cl=Calendar.getInstance();
                // get date current month
```

```
            int month=cl.get(Calendar.MONTH);
            if(month>=7 && month<=10)
                    return orderid+"  order is not approved";
            else
                    return orderid+" order is Approved";
        }//method
}//class
```

applicationContext.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-2.0.xsd">
<bean id="targetBean" class="com.nt.service.OrderApprover"/>
<bean id="auditingAdvice"  class="com.nt.aspect.AuditingAdvice"/>
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target" ref="targetBean"/>
<property name="interceptorNames">
<list>
<value>auditingAdvice</value>
</list>
</property>
</bean>
</beans>
```

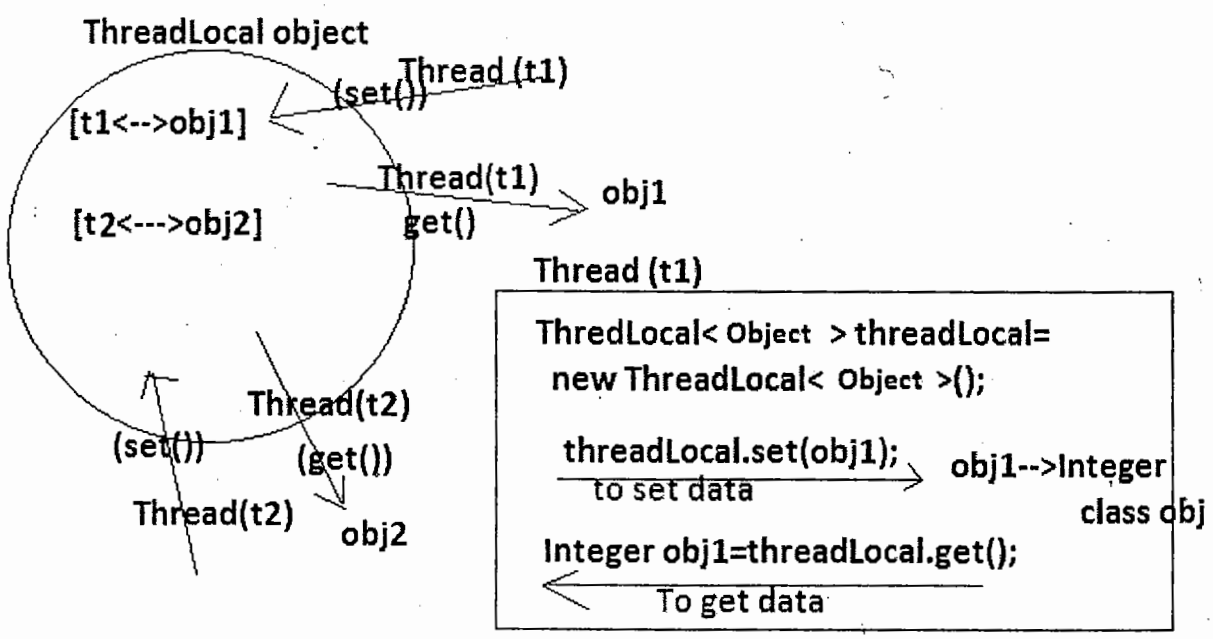
ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.OrderApprover;
public class ClientApp {
  public static void main(String[] args) {
          //create IOC container
          ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
          //get ~~Bean~~ Proxy object
          OrderApprover proxy=ctx.getBean("pfb",OrderApprover.class);
          //Call B.methods
          System.out.println("Order Approved?"+proxy.approveOrder(1234));
          System.out.println("-----------------------------------------------");
          System.out.println("Order Approved?"+proxy.approveOrder(-4567));
          System.out.println("-----------------------------------------------");
          System.out.println("Order Approved?"+proxy.approveOrder(0));
          System.out.println("-----------------------------------------------");
 }//main
}//class
```

- Security means authentication and authorizatrion. Checking identity of a user is called authentication.Checking the access permission is called authorization.Security is mainly authentication
- We need to authentication the user before entering into target method/business method. So it is recommended to develop secutiy logic as before advice logic.
- Since target method call cannot supply username, password everytime. We need to make client performing login/signin operation, submitting username, password before calling target method.These username, password must be saved temporarly to use then when Real B.Method/ target method is called for this we can use ThreadLocal object.
- If we want to make data specific to each thread having global visibility without using local variable and synchronization concept then we can keep that data in ThreadLocal object that is taken as instance variable.
- This ThreadLocal object allows each thread to keep one object and doesn't allow other threads to access and use that object. So we can say ThreadLocal object maintaining data in Thread scope.

In java we can keep data in 4 scopes.

1) Local
2) Instance
3) Class
4) ThreadLocal

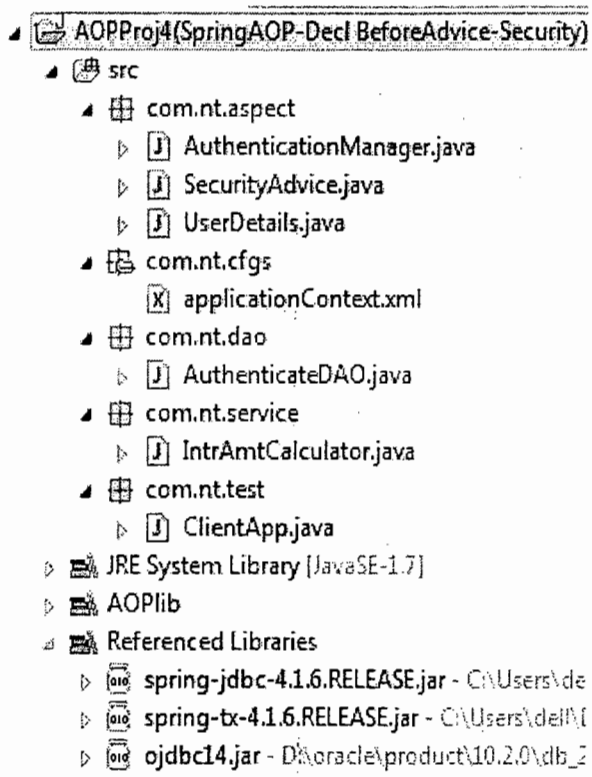


eg: userlist (database table in Oracle)

Username(VC2)	password(VC2)
Raja	Rani
King	Kingdom

Without using Synchronization Concept if you want to mentain thread safe data having global visibility use ThreadLocal object. The data placed by one thread in ThreadLocal object only by that thread.

Thread local object is internally a map collection having thread name as key, object as the value.While Operating ATM we give CardNo,PINno but they will not be used for authentication immediately.They will be actually used for authentication before performing Serious business operation like withdraw,deposite and etc...



AuthenticateDAO.java

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
import com.nt.aspect.UserDetails;
public class AuthenticateDAO {
	private static final String AUTH_QRY="select count(*) from userlist where uname=? and
pwd=?";
	private JdbcTemplate jt;
	public void setJt(JdbcTemplate jt) {
		this.jt = jt;
	}
	public boolean authenticate(UserDetails ud){
		int cnt=jt.queryForInt(AUTH_QRY, ud.getUser(),ud.getPwd());
		if(cnt==0)
			return false;
		else
			return true;
	}
}
```

AuthenticationManager.java

```
package com.nt.aspect;
import com.nt.dao.AuthenticateDAO;
public class AuthenticationManager {
	private ThreadLocal<UserDetails> tl=new ThreadLocal<UserDetails>();
	private AuthenticateDAO dao;
		public void setDao(AuthenticateDAO dao) {
		this.dao = dao;
	}

	public void signIn(String user,String pwd){
		UserDetails ud=new UserDetails(user,pwd);
		tl.set(ud);
	}
	public void signOut(){
		tl.remove();
	}
	public boolean isAuthenticated(){
		//get UserDetails object
		UserDetails ud=tl.get();
		boolean flag=dao.authenticate(ud);
		return flag;
	}
}
```

SecurityAdvice.java

```
package com.nt.aspect;
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;
public class SecurityAdvice implements MethodBeforeAdvice{
	private AuthenticationManager am;
	public void setAm(AuthenticationManager am) {
		this.am = am;
	}
	@Override
	public void before(Method method, Object[] args, Object target)
			throws Throwable {
		//perform Authentication
			boolean flag=am.isAuthenticated();
		if(flag==false)
			throw new IllegalArgumentException("In Valid Credentials");
	}
}
```

UserDetails.java

```
package com.nt.aspect;
public class UserDetails {
	private String user,pwd;
	public String getUser() {
```

```
            return user;
        }
        public void setUser(String user) {
            this.user = user;
        }
        public String getPwd() {
            return pwd;
        }
        public void setPwd(String pwd) {
            this.pwd = pwd;
        }
        public UserDetails() {
        }
        public UserDetails(String user, String pwd) {
          this.user = user;
            this.pwd = pwd;
        }
}
```

applicationContext.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-2.0.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="authDAO" class="com.nt.dao.AuthenticateDAO">
<property name="jt" ref="template"/>
</bean>
<bean id="authManager" class="com.nt.aspect.AuthenticationManager">
<property name="dao" ref="authDAO"/>
</bean>
<bean id="targetBean"  class="com.nt.service.IntrAmtCalculator"/>
<bean id="securityAdvice" class="com.nt.aspect.SecurityAdvice">
<property name="am" ref="authManager"/>
</bean>
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target" ref="targetBean"/>
<property name="interceptorNames">
<list>
<value>securityAdvice</value>
</list>
```

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

IntrAmtCalculator.java

```
package com.nt.service;
public class IntrAmtCalculator {
        public float calcIntrAmt(float p,float t,float r){
                return p*t*r/100.0f;
        }
}
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.aspect.AuthenticationManager;
import com.nt.service.IntrAmtCalculator;
public class ClientApp {
        public static void main(String[] args) throws Exception{
                //Create IOC Container
                ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                // get AuthtenticationManager object
                AuthenticationManager
authManager=ctx.getBean("authManager",AuthenticationManager.class);
                 //signIN
                authManager.signIn("raja","rani1");
                 //call b.method
                 IntrAmtCalculator proxy=ctx.getBean("pfb",IntrAmtCalculator.class);
        float amt=proxy.calcIntrAmt(20000,10,2);
        System.out.println("Intr Amt="+amt);
        //signOUT
        authManager.signOut();
         }//main
}//class
```

Jar: AOP+DAO+IOC

AfterAdvice/AfterReturningAdvice:

- This advice method/Logic executes once target method finishes the execution and before it returns the control to caller method.
- This advice can catch the return value but cannot modify the return value. To develop this advice we need to take a class implementing AfterReturningAdvice(i) and should provide implementation for afterReturning(-,-,-,-) method.
- Public void afterReturing(Object retVal,Method method,Object args[],Object target)

Usecase:

i) Generating discount voucher for next purchage based on current purchage bill amount.

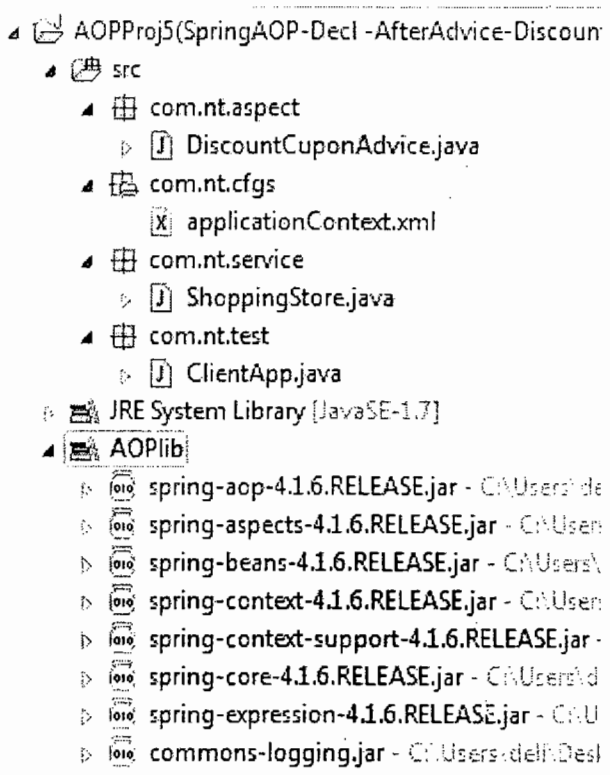
ii) ATM pin generation.

Important/Control Points:

a) We can access target method arguments but there is no use of modifying them because control comes to target method only after finishing execution of target method.
b) Cannot control target method execution because control comes to advice method only after executing target method. But by throwing exception we can stop target method sending return value to caller method and verification.
c) Can access the return value but cannot modify return value if modify that will not reflect.

AOP is designed based on decorator, proxy design patterns.

Note:
Example application on AfterAdvice to generate Discount Coupan based OnBillAmount.



DiscountCuponAdvice.java

```
package com.nt.aspect;
import java.io.FileWriter;
import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;
public class DiscountCuponAdvice implements AfterReturningAdvice {
        @Override
        public void afterReturning(Object returnValue, Method method, Object[] args,
                        Object target) throws Throwable {
                                float billAmt=(Float)returnValue;
                //we can not modify return Value.
                billAmt=billAmt+100;
                returnValue=billAmt;
                                String msg=null;
                if(billAmt>=50000)
                        msg="Avail 30% Discount on next purchase";
```

```
            else if(billAmt>=30000)
                    msg="Avail 20% Discount on next purchase";
            else
                    msg=" Avail 10% Discount on next purchase";

            //store discount cupon in a file
            FileWriter fw=new FileWriter("D:\\cupon.Tt");
            fw.write(msg);
            fw.flush();
            fw.close();
            }//afterReturning
}//class
```

ShoppingStore.java

```
package com.nt.service;
public class ShoppingStore {
        public float calcBillAmt(String item,float price,int qty){
                return price*qty;
        }
}
```

applicationContext.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.xsd">
<bean id="targetBean"  class="com.nt.service.ShoppingStore"/>
<bean id="discountAdvice" class="com.nt.aspect.DiscountCuponAdvice"/>
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target" ref="targetBean"/>
<property name="interceptorNames">
<list>
<value>discountAdvice</value>
</list>
</property></bean>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.ShoppingStore;
public class ClientApp {
        public static void main(String[] args) {
                // create IOC Container
                ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
    //get Proxy object
                ShoppingStore proxy=ctx.getBean("pfb",ShoppingStore.class);
```

```
            //call target method
            float billAmt=proxy.calcBillAmt("table", 3000,20);
            System.out.println("Bill Amount="+billAmt);
        }
 } //main& Class
```

We can use after advice to check weather the return value generated by target method is valid value or not. If not valid we can throw exception in Afteradvice and we can stop that return value being return to ClientApp (caller).

Usecase:
If target method returns ATM pin we can check weather that generated pin is weak pin or strong pin in Afteradvice. If it is a weak pin then we

- AOPProj6(SpringAOP-Decl AfterAdvice-PinVerifier)
  - src
    - com.nt.aspect
      - CheckPinAdvice.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.service
      - PinGenerator.java
    - com.nt.test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]
  - AOPlib
  - Referenced Libraries

CheckPinAdvice.java

```
package com.nt.aspect;
import java.lang.reflect.Method;
import java.util.Arrays;
import org.springframework.aop.AfterReturningAdvice;
public class CheckPinAdvice implements AfterReturningAdvice {
        @Override
        public void afterReturning(Object retValue, Method method, Object[] args,
                        Object target) throws Throwable {
            System.out.println("Generated pin"+retValue);
            System.out.println("target class"+target.getClass());
            System.out.println("target method"+method.getName());
            System.out.println("target method args"+Arrays.toString(args));
          int cardNo=(Integer)args[0]+100;
                        //get return value
            int pin=(Integer)retValue;
            //check the pin
            if(pin<=999)
                        throw new IllegalArgumentException(" Weak Pin generated");
        }//aterReturning
}//class
```

PinGenerator.java

```
package com.nt.service;
import java.util.Random;
public class PinGenerator {
	public int  generatePin(int cardNo){
		//generate ATM Pin
		Random rad=new Random();
		int pin =rad.nextInt(9999);
		return pin;
	}
}
```

applicationContext.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.xsd">
<bean id="targetBean"  class="com.nt.service.PinGenerator"/>
<bean id="pinAdvice" class="com.nt.aspect.CheckPinAdvice"/>
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target" ref="targetBean"/>
<property name="interceptorNames">
<list>
<value>pinAdvice</value>
</list>
</property>
</bean>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.PinGenerator;
public class ClientApp {
	public static void main(String[] args) {
		// create IOC Container
		ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
	//get Proxy object
		PinGenerator proxy=ctx .getBean("pfb",PinGenerator.class);
		//call target method
		int pin=proxy.generatePin(88888);
		System.out.println("Generated Pin="+pin);
	}//main
}//class
```

ThrowsAdvice:

- This advice will execute when exception is raised in target method. Using this advice we cannot catch and handle the exception, but we can see the details of exception that is raised.
- We can use this advice to develop common/central Exception logger that logs all the exception related messages that are raised in various target methods of target class.
- We can catch and handle the exception raised by target method in client Application where target method will be called.
- To develop this advice we need to take class that implements "ThrowsAdvice (i)" (marker interface--->Empty interface) and we need to place one or more afterThrowing methods having the following signatures.
- *public void afterThrowing (<sub class of Throwable>)*

Or

- *public void afterThrowing (Method method, Object args[], Object target <subclass of throwable>)*

Control points/important points:

a) We can access Target Method params, but there is no use of modifying them.
b) We can not control target method execution because control comes to throwsAdvice after executing target methods that to for exception raising.
c) We cannot catch the return value of target method.

NOTE

Throws advice logic not only executes when exception is raised in target method. It also respond for the exception raised in other advises configured on the target method...

1) To develop common/central exception logger that locks all exception related messages(useful in incident management)

2) To cascade technology specific exceptions to project specific custom exceptions (to achieve loose coupling between Presentation tier and Business tier component)

Note: servlet, jsp, html etc...files are called view, controller components (presentation tier component). Dao and service classes are called bisiness tier components. If business tier components propagate their technology specific exception to presentation tier component their tight coupling problem will be raise. To solve this problem use ThrowsAdvice to translate technology exception to project specific custom exception.

- AOPProj7(Spring AOP Decl-ThrowAdvice-Exceptior
  - src
    - com.nt.aspect
      - ExceptionLoggerAdvice.java
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.service
      - PriceCalculator.java
    - com.nt.test
      - ClientApp.java
  - JRE System Library
  - AOPlib

ExceptionLoggerAdvice.java

```
package com.nt.aspect;
import java.lang.reflect.Method;
import java.util.Arrays;
import org.springframework.aop.ThrowsAdvice;
public class ExceptionLoggerAdvice implements ThrowsAdvice {
	public void afterThrowing(IllegalArgumentException ie){
		throw new NumberFormatException("Changed exception");
	}
	public void afterThrowing(Method method,Object args[],Object target,Throwable ie){
	System.out.println(ie.getClass()+" Exception is raised in"+method.getName()+" with args
"+Arrays.toString(args)+" in "+target.getClass());
	}
}
```

PriceCalculator.java

```
package com.nt.service;
public class PriceCalculator {
	public float calcPrice(float price,float qty){
		if(price==0.0f || qty==0.0f)
			throw new IllegalArgumentException(" price and qty must not be zero..");
		return price*qty; }
}
```

applicationContext.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.xsd">
<bean id="targetBean"  class="com.nt.service.PriceCalculator"/>
<bean id="throwsAdvice" class="com.nt.aspect.ExceptionLoggerAdvice"/>
<bean id="pfb"  class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="target" ref="targetBean"/>
<property name="interceptorNames">
<list>
<value>throwsAdvice</value>
</list>
</property>
</bean>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.PriceCalculator;
public class ClientApp {
	public static void main(String[] args) {
		// create IOC Container
```

```
            ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
    //get Proxy object
            PriceCalculator proxy=ctx.getBean("pfb",PriceCalculator.class);
            //call target method
            try{
            float price=proxy.calcPrice(45.65f,0);
            System.out.println("Price="+price);
            }
            catch(Exception ie){
                    System.out.println("Internal Problem"+ie.getClass()+"
"+ie.getMessage());
            }
        }//main
}//class
```

Jars: same

- ⇨ If exception is raised in target method, control comes to throws advice and looks for afterThrowing methods in the following order.

  1) Exact Exception type
  2) Nearest super class raised Exception type
  3) Super class raised Exception type.

more vs Less -----: Less
Super Vs exact -----: exact
Super Vs Near super -----: Near Super
Less Vs Exact -----:

Note: If multiple afterThrowing method contains exact match the one params method executes.

Note: If multiple afterThrowing method contains super class of raised exception then one params method executes.

Note: If 1 afterThrowing() contains exact matching and another afterThrowing() contains super type then the exact match afterThrowing will execute. If both different then nearest superclass param contained method will execute. if clash comes between one param, more param methods having same exception time then priority given to one param.

- ⇨ We can also use ThrowsAdvice to graph/cascade/Translate one form of Exception to another form of Exception. LikeTranslating the business tier component generated technology specific exception to application specific exception.

Advice Type	When advice logic executes	Access to target method params	Access to target method return value	Control to target method execution
Aroud Advice	Aroubd target method execution	Can see and modify	Can see and modify	Yes (USECASE:Cacheing,Around logging,Performance monitoring)

Before Advice	Before target method Execution	Can see and modify	NA	NA (can stop by throwing Exception) (USECASE: Auditing,SecurityCheck)
After Advice	After target method execution	Can see but not modify	Can see but not modify	NA (USECASE:GiftVoucher generator,ATM Pin generator)
Throws Advice	When exception is raised in target method	Can see but not modify	NA	Na (USECASE:CommonException logger ,Exception Translator)

Programatic Spring AOP:

- ⇨ Here no xml file support or configuration file support will be taken to generate proxy class by supplying Target class, advice class.
- ⇨ All inputs will be given directly in client application through java statement.

This is not recommended approach.

Here we use ProxyFactory class directly in client app to generate the proxy class.

```
 ProxyFactory factory= new ProxyFactory();
factory.setTarget ();
factory.addAdvice();
proxy=factory.getProxy();
proxy.bm1();
proxy.bm2();   //call B.methods
```

- ⇨ *Programatic SpringAO is not recommended to use because it makes the programmer to modify java code to enable or disable Aspects and doesnot allow programmer to take the benefits of IocContainer.*

PointCut:

- ⇨ So far the aspects that we are developing will be applied on all the methods of target class. In some situations we don't want to apply aspects on certain joinpoints or methods of target class.
- ⇨ For this we can write additional conditional logic in advice class to execute advice logic only for specific joinPoint methods (Target methods)

```
public class PerformanceMonitoringAdvice implements MethodInterceptor{

    public Object invoke(MethodInvocation invocation) throws Throwable {

        String methodName=null;

methodName=invocation.grtMethod().grtName();
```

e

```
    if(methodName.equalse("add")){

    //performance monitpring logic

        }
else{
return invocation.proceed();
}
   }
}
```

- In the above approach the generated proxy class overrides all the methods of target class having logic to apply aspects on target method.But advice logic will be executed only for specific method because of ther condition that is specified. This indicates proxy class is not generated in optimized manner because it is also overriding those target methods on which we don't want to apply aspects moreever those methods of target class should be executed directly without touching proxy class when class Then they are called from clients.
- To overcome the above problem use pointcuts. Pointcut is a collection of join points on whom you want to advice the aspects. It specifies which aspects has to be applied on which joinPoints.

Standa~~lone~~ derd procedure to work with poincuts(declarative)

a) Configure user defined or predefined pointcut class as spring bean

```
<bean id="nmp"  class="pkg.NamedMatchMethodPointcut">  predefine
        <property name="mappedName" value="add"/>
        </bean>
```

b) Configure Advisor class as spring bean

```
        <bean id="advisor"  class="pkg.DefaultPointcutAdvisor">
        <property name="advice" value="pmAdvice"/>
        <property name="pointcut" value="nmp"/>
        </bean>
```

Note: we can directly configure "NamedMatchMethodPointcutAdvisor to avoid step(a).

c) Configure ProxyFactoryBean

```
        <bean id="pfb"  class="pkg.ProxyFactoryBean">
        <property name="target" value="------"/>
        <property name="interceptorNames" >
        <list>
        <value>advisor</value>
        </list>
        </property>
        </bean>
```

d)-Get Proxy object i client App

```
<Target> proxy;
  proxy=ctx.getBean("pfb",Target.class)
```

There are 2 types of pointcuts.

1) static Pointcut:

- Allows to specify target class name, method names while preparing the Pointcut. But that can't specify target methods argument values. This indicates static Pointcuts will be processed during weaving process that generates proxy class.

2) Dynamic Pointcut:

- This allows to specify target class, target method names and their arguments values. That means this pointcut will be processed after target method call on proxy object dynamically at runtime.

eg: Apply aspect on "add" method is static pointcut.

Apply aspect on "add" method when argument values are <100 is dynamic pointcut.

- While working with pointcut the generated proxy class override only those methods that are specified in pointcut to apply aspects. Due to this proxy class code will be optimized to improve performance.
- To develop userdefined pointcut we need to take a class implementing pointcut Interface. Spring api has supplied some predefined classes implementing this interface.
- They are :- 3 PREDEFINE STATIC POINTCUT CLASSES:-
  - StaticMethodMatcherPointcut (AC)
  - NameMatchMethodPointcut (C)
  - JdkRegexpMethodPointcut (C)

Dynamic pointcut

1) DynamicMethodMatcherPointcut

- To develop custom pointcut classes extend from above classes instead of implementing Pointcut Pointcut Interface.Advisor is the combination of Pointcut+Aspect/advice.
- In order to apply aspects specifying pointcuts we take the support of advisor. Every advisor class is the implementation class of Advisor (i)
- The spring API supplied predefined Advisor are
  - DefaultPointcutAdvisor
  - NameMatchMethodPointcutAdvisor and etc..
- While working with static pointcut classes we need to override matches method that gives access to target class and target method. If this method returns true for certain target method aspect will be applied for that method, otherwise it will not be applied.
- Inorder to develop dynamic pointcut we need to take a class extending from DynamicMethodMatcherPointcut class and implements matches (Method method, Object [] args, class<?> target) method.If this method returns true for the certain target method then aspect will be applied on that method otherwise aspect will not be applied.
- Instead of developing separate userdefined class by extending from abstract class StaticMethodMatcherPointcut or DynamicMethodMatcherPointcut we can use spring API supplied concrete static pointcut class NameMatchMethodPointcut as shown below:

```
NameMatchMethodPointcut nmp=nerw NameMatchMethodPointcut();
nmp.setMappedNames("Sum", "null");
DefaultPointcutAdvisor advisor=new DefaultPointcutAdvisor(nmp, new
PerformanceMonitoringAdvice()));
factory.addAdvisor(advisor);
```

Working with Pointcut in declarative spring AOP:

In this approach we can configure pointcut classes, advisor classes, Adviseclassesas beans in configuration file and we can specify advisor directly as the values of "interceptorNames" property while configuring ProxyFactoryBean.

Instead of working with "NameMatchMethodPointcut" "DefaultPointcutAdvisor" classes separately we can use "NameMatchMethodPointcutAdvisor" class directly to bring the same effect.

Stadard procedure to work with Poincuts(programmatic):

a) Create object for userdefined or predefined poincuts

```
NameMatchMethodPointcut nmp=nerw NameMatchMethodPointcut();
nmp.setMappedNames("add");
```

b) Create Advice class object

```
PerformanceMonitoring  pmAdvice= new PerformanceMonitoring();
```

c) Create Advisor class object

```
DefaultPointcutAdvisor advisor=new DefaultPointcutAdvisor(nmp, pmAdvice);
```

Note: we can directly use "NameMatchMethodPointcutAdvisor" to avoid step (a).

~~Step a)~~ Example:

```
NameMatchMethodPointcutAdvisor  advisor=nerw
NameMatchMethodPointcutAdvisor();
advisor.setMethodName("add");
    advisor.addAdvice(pmAdvice);
```

d) ~~step b)~~add Advisor ProxyFactory object

<Target class> proxy=factory.getProxy();

```
ProxyFactory factory= new ProxyFactory();
factory.setTarget(----);
factory.addAdvisor(advisor);
```

Note:- there are abstract,concrete static pointcut classes given by spring aop api so we can creates static pointcuts without developing user define classes .But while working with dynamic pointcut classes this provision is not provided. we should always develop user define dynamic pointcut classes.

eg: Don't take pointcut classes specific to one advice because same pointcut class can be used for different advises.

AOPProj9(Spring AOP Programatic when pointcut advises are not applied generally all advises will be apply on all target methods of target class. to overcome this problem we designed and used to pointcut through pointcut advises

- src
  - com.nt.aspect
    - CacheAdvice.java
    - LogAroundAdvice.java
    - PerformanceMonitoringAdvice.java
  - com.nt.service
    - ArithmeticService.java
  - com.nt.test
    - ClientApp.java
- JRE System Library [JavaSE-1.7]
- AOPlib

CacheAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class CacheAdvice implements MethodInterceptor {
        Map <String,Object> cache=new HashMap<String,Object>();
                @Override
        public Object invoke(MethodInvocation invocation) throws Throwable {
      String
key=invocation.getMethod().getName()+Arrays.toString(invocation.getArguments());
            Object retValue=null;
       if(!cache.containsKey(key)){
          retValue=invocation.proceed();
          cache.put(key, retValue);
          System.out.println("From target method");
          return retValue;
       }
       retValue=cache.get(key);
       System.out.println("From cache");
       return retValue;
        }
}
```

PerformanceMonitoringAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class PerformanceMonitoringAdvice implements MethodInterceptor {
  long start,end;
        @Override
        public Object invoke(MethodInvocation invocation) throws Throwable {
                start=System.currentTimeMillis();
                 Object retVal=invocation.proceed();
                end=System.currentTimeMillis();
                System.out.println(invocation.getMethod().getName()+" with
args"+Arrays.toString(invocation.getArguments())+" has taken "+(end-start)+"ms");
                return retVal;
                }
}
```

CommonDynaPointcut.java

```
package com.nt.poincut;
import java.lang.reflect.Method;
import org.springframework.aop.support.DynamicMethodMatcherPointcut;
import com.nt.service.ArithmeticService;
```

```
public class CommonDynaPointcut extends DynamicMethodMatcherPointcut {
        @Override
        public boolean matches(Method method, Class<?> target, Object[] args) {
                if(target==ArithmeticService.class && method.getName().equals("mul") &&
(Integer)args[0]>=1000 &&(Integer)args[1]>=1000)
                  return true;
                else
                        return false;
                }
}
```

PerformanceMonitoringPointcut.java [Static pointcut]

```
package com.nt.poincut;
import java.lang.reflect.Method;
import org.springframework.aop.support.StaticMethodMatcherPointcut;
import com.nt.service.ArithmeticService;
public class PerformanceMonitoringPointcut extends StaticMethodMatcherPointcut {
        @Override
        public boolean matches(Method method, Class<?> target) {
                if(target==ArithmeticService.class  &&  method.getName().equals("add"))
                        return true;
                else
                        return false;
                }
}
```

ArithmeticService.java [Target class]

```
package com.nt.service;
public class ArithmeticService {
        public int add(int x,int y){
                return x+y;
        }
        public  int sub(int x,int y){
                return x-y;
        }
        public int mul(int x,int y){
                return x*y;
        }
}
```

ClientApp.java [programmatic/applying pointcut/static/dynamic]

```
package test;
import org.springframework.aop.framework.ProxyFactory;
import org.springframework.aop.support.DefaultPointcutAdvisor;
import org.springframework.aop.support.NameMatchMethodPointcut;
import com.nt.aspect.CacheAdvice;
import com.nt.service.ArithmeticService;
public class ClientApp {
        public static void main(String[] args) {
```

```
            //Get ProxyFactory class object
            ProxyFactory pf=new ProxyFactory();
            //set Target class object
            pf.setTarget(new ArithmeticService());
      /*    //set Advice class object
            pf.addAdvice(new PerformanceMonitoringAdvice()); */
/*          //prepre advisor (poincut+ advice/aspect)
            DefaultPointcutAdvisor advisor=new DefaultPointcutAdvisor(new
PerformanceMonitoringPointcut();
                  new PerformanceMonitoringAdvice());
            pf.addAdvisor(advisor); */
      /*     DefaultPointcutAdvisor  PMadvisor=new DefaultPointcutAdvisor(new
CommonDynaPointcut()
new PerformanceMonitoringAdvice());
                        DefaultPointcutAdvisor  Cacheadvisor=new
DefaultPointcutAdvisor(new CommonDynaPointcut(),
                  new CacheAdvice());
    pf.addAdvisor(0,Cacheadvisor);
    pf.addAdvisor(1, PMadvisor); */
      /*     DefaultPointcutAdvisor  PMadvisor=new DefaultPointcutAdvisor(new
PerformanceMonitoringPointcut(),
                new PerformanceMonitoringAdvice());
     DefaultPointcutAdvisor  Cacheadvisor=new DefaultPointcutAdvisor(new
CommonDynaPointcut(),
     new CacheAdvice());
     pf.addAdvisor(0,PMadvisor);
     pf.addAdvisor(1,Cacheadvisor); */
                        NameMatchMethodPointcut nmp=new
NameMatchMethodPointcut();
            nmp.setMappedNames("add","mul");
            DefaultPointcutAdvisor advisor=new DefaultPointcutAdvisor(nmp,new
CacheAdvice());
            pf.addAdvisor(advisor);
            //get Proxy object
            ArithmeticService proxy=(ArithmeticService)pf.getProxy();
            // Call target method
            System.out.println("Sum="+proxy.add(1000,2000));
            System.out.println("-----------------------------------");
            System.out.println("Sub="+proxy.sub(100000,20000));
            System.out.println("-----------------------------------");
            System.out.println("Mul="+proxy.mul(10000,20000));
            System.out.println("-----------------------------------");
            System.out.println("Mul="+proxy.mul(10000,20000));
            System.out.println("-----------------------------------");
            System.out.println("Mul="+proxy.mul(100,200));
            System.out.println("-----------------------------------");
      }//main
}//class
```

- Use spring AOP programmatic/declarative environment we can work with pointcuts in 3 ways

1) Develop user defined pointcut class and inject to Advisor.
2) Use spring API supplied predefined concrete class and inject to Advisor
3) Use spring API supplied predefined Advisor class. Which also internally maintain pointcut effect.

- With respect to SpringAOP declarative pointcuts. We can develop static Pointcut either as usedefined class extending from StaticMethodMatcherPointcut or we can use spring specified direct concrete classes like NameMatchMethodPointcut.
- Inorder to work with DynamicMethodPointcut we must take user defined class extending from DynamicMethodMatcherPointcut and there is no direct concrete classes supplied by spring API to work with dynamic pointcut.

Aspect JAOP:

- A third party AOP framework where we can develop aspect classes as pojo classes without having tight coupling with spring API. i.e aspect class need not implement spring api interfacesand need not to extend spring API class.
- But we need to use aspectJ api inside the aspects classes. Different between spring AOP and AspectJ AOP.

Spring AOP	Plain Aspectj AOP
Performs runtime weaving	=> Performs compile time weaving
Generates proxy class at runtime	=> Generates proxy class at compile time. Use AspectJ compiler.
Supports only methods as jionPoints.	=>Supports fields, constructors methods as joinpoints.
Support both static and dynamic pointcut	=>Supports only static Pointcuts.
Doesn't allow annotations	=> Allows annotations.
Can be used in programmatic, Decalarative xml approach.	=> can be used in declarative (xml) and annotations.
Advice classes are non POJO classes.	=> Advice classes are POJO classes.

- The limitations with Spring AOP is the Advice classes are invasive classes (should implements or extends from Spring API interface or classe).
- From spring 2.0 the AspectJ AOP is integrated wih spring AOP due to this limitations are applied on AspectJAOP
  - Supports only method level joinpoints.
  - Performs runtime weaving i.e proxy class will be generated at runtime.
  - Still supports only static pointcuts.
  - No AspectJ compiler is required.

We can use spring integrated AspectJ AOP in 2 ways

a) Declarative AspectJ AOP (using xml)

b) Annotations AspectJ AOP

Note: AspectsJAOP also supports 4 types of Advices

Declarative AspectsJAOP:

- Here we develop aspect classes as pojo classes by using aspectJ api internally and we configure them in xml file by using tags as aspect classes.We need to import AOP namespace for thisconfiguration.
- The important tags are

<aop: config>--->to enable AspectJ AOP from spring container.
<aop: aspect>--->to confg spring bean as Aspect class.
<aop: around>----->to configure aspect class method as Around advice.
<aop: before>----->to configure aspect class method as before advice
<aop: after-returning>---> to configure aspect class method as after advice.
<aop: after-throwing>----->to configure aspect class method s throws advice
<aop: pointcut>---->to configure pointcut

We use OGNL for pointcut expression

Syntax:

execution (<return type><package name>.<class name>.<method name> (..))

Around Advice:

- This advice executes around target methods.
- The use cases are caching, logging, performanceMonitoring and etc....
- To develop this advice we can use any orbitary class having any arbitrary method with following signature.

public Object <method>(ProceedingJoinPoint pjp) throws Throwable

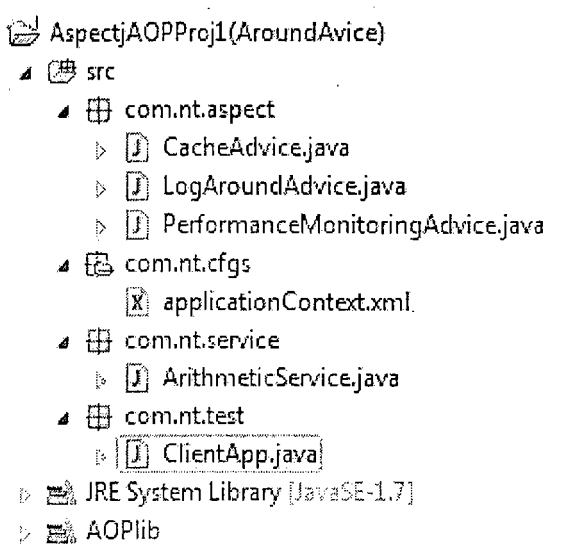
ProcedingJoinPoint pjp can be used to get target class, method info and to control target method invocation.

Note:- in cfgs.xml
It is always recomanded to have separate pointcut for every Aspect/Advice without having dependency with other Aspects/Advices.

Steps to work with AspectJ AOP:

1) Develop and configure target class as singleton.
2) Develop and configure aspect class as singleton.
3) Enable AspectJ AOP using <aop:config> tag
4) Prepare pointcut expression using <aop:pointcut> tag
5) Config Aspect/Advice spring bean as AspectJ aspect class using <aop: aspect> tag and also configure it's methods as advice methods.

eg:



+ aspectjrt-1.5.3.jar,aspectjweaver-1.5.3.jar (use 1.8 <version>

CacheAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import java.util.HashMap;
import org.aspectj.lang.ProceedingJoinPoint;
public class CacheAdvice {
        HashMap<String,Object>  cache=new HashMap<String,Object>();
      public Object  caching(ProceedingJoinPoint pjp)throws Throwable{
            String key=pjp.getSignature()+Arrays.toString(pjp.getArgs());
                  if(!cache.containsKey(key)){
                  Object retVal=pjp.proceed();
                  cache.put(key,retVal);
                  System.out.println("From taget method");
                  return retVal;
            }
            else{
                  System.out.println("From cache");
                  return cache.get(key);
            }
      }//caching
}//class
```

LogAroundAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aspectj.lang.ProceedingJoinPoint;
public class LogAroundAdvice {
      public Object logging(ProceedingJoinPoint pjp)throws Throwable{
            System.out.println("Entering into "+pjp.getSignature()+" with
args"+Arrays.toString(pjp.getArgs()));
            //Get args and modify args
            Object args[]=pjp.getArgs();
            if((Float)args[0]<=50000){
```

```
                        args[2]=(Float)args[2]-1;
                }
                        Object retVal=null;
                if((Float)args[0]==0 ||(Float)args[1]==0 ||(Float)args[2]==0 )
                        throw new IllegalArgumentException("Provide +ve values as the p,t,r
values");
                else
                        retVal=pjp.proceed(args);
                // add Service tax on intr amt
                retVal=(Float)retVal+(Float)retVal*0.05f;
        System.out.println("Exiting from "+pjp.getSignature()+" with
args"+Arrays.toString(pjp.getArgs()));
        return retVal;
          }//method
}//class
```

PerformanceMonitoringAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aspectj.lang.ProceedingJoinPoint;
public class PerformanceMonitoringAdvice {
   long start,end;
        public Object pMonitor(ProceedingJoinPoint pjp)throws Throwable{
                start=System.currentTimeMillis();
                Object retVal=pjp.proceed();
                end=System.currentTimeMillis();
                System.out.println(pjp.getSignature()+" with"
+Arrays.toString(pjp.getArgs())+" has taken "+(end-start)+"ms");
                return retVal;
                }
}
```

IntrAmtCalculator.java

```
package com.nt.service;
public class IntrAmtCalculator {
        public float  calcSimpleIntrAmt(float p,float t,float r){
                return (p*t*r)/100.0f;
        }
        public float calcCompoundIntrAmt(float p,float t,float r){
                return (float) (p*Math.pow((1+r/100.0f),t)-p);
        }
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
<bean id="target" class="com.nt.service.IntrAmtCalculator"/>
<bean id="logAdvice" class="com.nt.aspect.LogAroundAdvice"/>
<bean id="cacheAdvice" class="com.nt.aspect.CacheAdvice"/>
<bean id="PMAdvice" class="com.nt.aspect.PerformanceMonitoringAdvice"/>
<aop:config>
<aop:pointcut id="ptc1"  expression="execution(* com.nt.service.*.*(..))"/>
<aop:pointcut id="ptc2"  expression="execution(
*com.nt.service.IntrAmtCalculatorInterface.calcCompoundIntrAmt(..))"/>
<aop:aspect id="logAspect" order="3" ref="logAdvice"  >
<aop:around  pointcut-ref="ptc1"  method="logging" />
</aop:aspect>
<aop:aspect id="cacheAspect"  order="1" ref="cacheAdvice">
<aop:around pointcut-ref="ptc2" method="caching"/>
</aop:aspect>
<aop:aspect id="PMAspect"  order="2" ref="PMAdvice">
<aop:around pointcut-ref="ptc1" method="pMonitor"/>
</aop:aspect>
</aop:config>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.IntrAmtCalculator;
public class ClientApp {
        public static void main(String[] args) {
                //create IOC container
                ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //GEt Proxy object
                IntrAmtCalculator proxy=ctx .getBean("target",IntrAmtCalculator.class);
                System.out.println(proxy.getClass());
                //Call target method
                System.out.println("Simple Intr Amt"+proxy.calcSimpleIntrAmt(10000,20,2));
        System.out.println("-----------------------------------------");
        System.out.println("Compound Intr Amt"+proxy.calcCompoundIntrAmt(60000, 20, 2));
                System.out.println("-----------------------------------------");
        System.out.println("Compound Intr Amt"+proxy.calcCompoundIntrAmt(60000, 20, 2));
        }
}
```

Jar: *AOPLib+aspectjrt1.5.3.jar+aspectweaver-1.5.3.jar* ***(use 1.8 <version>)***

Note:

- While working with AspectJ AOP AroundAdvice the modification done in target method args in Around advice will be reflected only when we callpjp.proceed (args) having that object[] as arugments. Here Object[] holds the modified argument values.

- ⇨ When we run the ClientApp of spring intrigated AspectJAOP the internally created IOC container maintains inmemory metadata along with AOP configurations based on <aop:config>and its sub tags.

- ⇨ When ctx.getBean(-) method is called with target class bean id ,the IOC container doesn't return ~~create~~ target class object because of <aop:config> tag based configuration. It generates proxy class extending form target class and returns that proxy class object to the clientApplication.

- ⇨ Like spring AOP if we start working with proxyInterface (The interface that is implemented by target class) in AspectJAOP then the generated proxy class will come because of jdk libraries. For this we need to specify proxy interface name in OGNL expression of pointcut. In this AspectJAOP,if proxy interfaces are not specified (if target class is specified) the CGLIB libraries will be utilized to generate the proxy class.

- ⇨ While working with ~~AspectJAop~~ already configuring pointcut expression is mandatory /without proxy interface even though you want to apply the advices on all the target methods.

Before Advice:

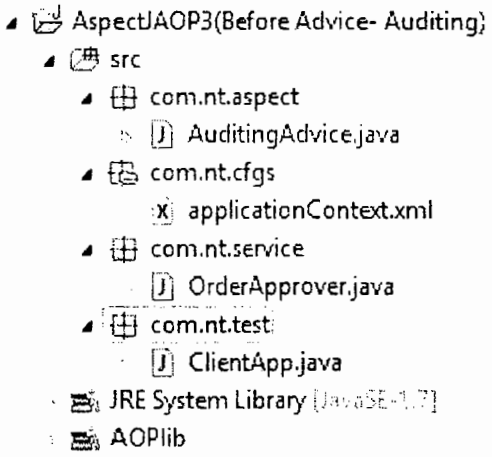
- ⇨ This advice executes before executing target method.
- ⇨ Using this advice we can access target method parameters but we can't control target method execution and we can't catch the return value.
- ⇨ Using this advice we can stop target method execution by throwing exception.
- ⇨ To develop this advice we can take any class having any method with following signature.

```
public void <Method> (JoinPoint jp)
or
public void <method> (JoinPoint jp, param1, param2,.....)
```

- ➢ To hold the target method argument values

- ➢ JoinPoint object just gives target method method details where as ProceedingJoinPoint object gives target method details, control on target method invocation.

- ➢ BeforeAdvice use cases are security check, auditing and etc



AuditingAdvice.java

```
package com.nt.aspect;
import java.io.FileWriter;
```

```
import java.util.Arrays;
import java.util.Date;
import org.aspectj.lang.JoinPoint;
public class AuditingAdvice {
	public void audit(JoinPoint jp, int oid)throws Throwable{
	System.out.println("target method:"+jp.getSignature());
		System.out.println("taget class "+jp.getTarget().getClass());
		System.out.println("target method args"+Arrays.toString(jp.getArgs()));
		System.out.println("Order Id is"+oid);
		Object args[]=jp.getArgs();
		args[0]=(Integer)args[0]+1;
		jp.getArgs()[0]=args[0];
		//peform auditing
		 FileWriter fw=new FileWriter("e:\\audit.log",true);
		 fw.write("order with order id "+oid+"  sumited for approval on Date"+new
Date());
		 fw.flush();
		 fw.close();
	}//audit
}//class
```

OrderApprover.java

```
package com.nt.service;
import java.util.Calendar;
public class OrderApprover {
	public String approve(int orderid){
		// get Current month
		Calendar cl=Calendar.getInstance();
		int h=cl.get(Calendar.MONTH);
		if(h>=7 && h<=10)
			return "We do not accept this "+orderid +"order in rainy seasno";
		else
			return orderid +" order is approved";
	}//approve
}//class
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
<bean id="orderApprover" class="com.nt.service.OrderApprover"/>
<bean id="auditAdvice"  class="com.nt.aspect.AuditingAdvice"/>
<aop:config>
<aop:pointcut id="ptc1" expression="execution(* com.nt.service.*.*(..)) and args(oid)"/>
<aop:aspect id="auditAspect"  ref="auditAdvice">
<aop:before method="audit" pointcut-ref="ptc1"/>
```

```
</aop:aspect>
</aop:config>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.OrderApprover;
public class ClientApp {
        public static void main(String[] args) {
                //create IOC container
                ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //get Proxy
                OrderApprover proxy=ctx .getBean("orderApprover",OrderApprover.class);
                System.out.println("status:"+proxy.approve(1001));
        }
}
```

- In AspectJAOP the beforeAdvice can access TargetMethod arguments but cannot modify their values. It is not similar to spring AOP before Advice.
- While working with beforeAdvice we can even get Targetmethod arguments without ~~making~~ taking the support of JoinPoint object. We need to take additional parameters support in advice method to store target method arguments and we must specify those parameters in pointcut expression.

In applicationContext.xml:

```
<aop: pointcut id="ptel" expression="execution(* com.nt.serivce.*.*(..)) and args (oid)"/>
```

While working with AspectJ AOP if we call cT.getBean(-) method by using the bean id whose class name is specified in pointcut OGNL expression then we get proxy class object. Otherwise we will get the given bean id related object.

AfterAdvice/After Returning Advice:

- This advice method executes when target method complete execution and before it returns the control/value to caller.
- To develop this advice we can take any class having the following signature method.

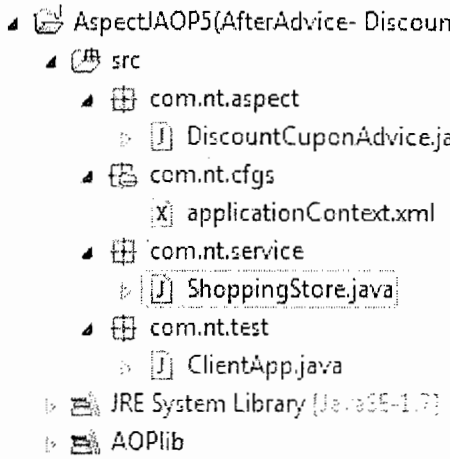
public void <method>(JoinPoint JP, <param to hold return value>)

Usecase:

1) Generating discountCoupn for next purchase.
2) ATM pin verifier.

- This advice allows to access target method args values. But doesn't allow to modify.
- This advice allows to access return value but doesn't allow to modify.
- By throwing exception in this advice we can stop return value being returned to caller method.
- The parameter taken in advice method to hold return value must be configured in <aop:after-returing> by using "returning" attribute.

```
<aop: after-returning method="coupon" pointcut-ref="ptct" returning="billAmt"/>
```



DiscountCuponAdvice.java

```
package com.nt.aspect;
import java.io.FileWriter;
import org.aspectj.lang.JoinPoint;
public class DiscountCuponAdvice {
	public void generateCupon(JoinPoint jp,float billAmt)throws Throwable{
		String cuponMsg=null;
		if(billAmt>=50000)
			cuponMsg="Avail 30% Discount on next purcharse";
		else if(billAmt>=30000)
			cuponMsg="Avail 20% Discount on next purchase";
		else
			cuponMsg="Avail 10% Discount on next purchase";
		  // Save Cupon in a file
		  FileWriter fw=new FileWriter("e:\\cupont.Tt");
		  fw.write(cuponMsg);
		  fw.flush();
		  fw.close();
	}//generateCupon
}//class
```

ShoppingStore.java

```
package com.nt.service;
public class ShoppingStore {
	public float generateBillAmt(String item,float price,float qty ){
		return price*qty;
	}
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
<bean id="shopping"  class="com.nt.service.ShoppingStore"/>
<bean id="discountAdvice" class="com.nt.aspect.DiscountCuponAdvice"/>
<aop:config>
<aop:pointcut  id="ptc1" expression="execution(* com.nt.service.*.*(..))"/>
<aop:aspect id="discountAspect"  ref="discountAdvice">
<aop:after-returning method="generateCupon" pointcut-ref="ptc1" returning="billAmt"/>
</aop:aspect>
</aop:config>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.ShoppingStore;
public class ClientApp {
        public static void main(String[] args) {
                //create IOC container
                ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //GEt Proxy object
                ShoppingStore proxy=ctx .getBean("shopping",ShoppingStore.class);
                //call B.method
                System.out.println("Bill Amt"+proxy.generateBillAmt("shirt",900,2));
        }
}
```

Throws Advice:

- This advice method executes when exception is raised in target method. This advice is useful to develop central, common exception logger.
- This advice doesn't catch and handle the exception, but allows to see the details of exception and translate/graph one form of exception to another form of exception.
- To develop this advice we can take a class having method with following signature.

```
public void <method name>(JoinPoint JP, <param of type subclass of
Throwable>);
```

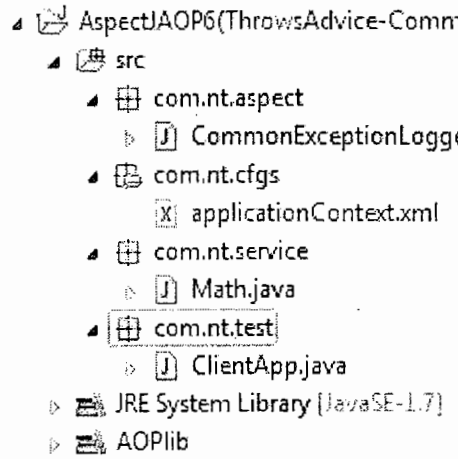
- The additional param of this advice method, that is capable of referring raised exception must be specified in <aop:after-throwing> tag by using "throwing" attribute.

eg: *<aop:after-throwing method="....." pointcut-ref="....." throwing="....."/>*

- Throws advice also responds for the exceptions that are raised in other advices that are applied on the target method.Throws advice is really usefull in real time to graoh or translate the technology specific exceptions of service,DAO classes into application specific Exception.This allows to keep presentation tier components untouchable even though service,DAO class technology are changed...

AspectJAOP (ThrowsAdivce-CommonExceptionLogger)

- ThrowsAdvice even useful for exception graphing, which is nothing but transforming of one form of exception to another form.
- While working with AspectJAOP specifying pointcut expression is mandatory even though you want to apply all the advices,on all the target methods of target class because if target class name or if implemented proxy interface is not there in the OGNL Expression of pointcut Expression the proxy class will not be generated when ctx.getBean() is called...



CommonExceptionLogger.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aspectj.lang.JoinPoint;
public class CommonExceptionLogger {
	public void exceptionLogging(JoinPoint jp,ArithmeticException ae){
		System.out.println(ae.getClass()+"Exception raised With Msg"
				+ae.getMessage()+" in  "+jp.getSignature()
				+"with args"+Arrays.toString(jp.getArgs()));
		throw new IllegalArgumentException("Provide valid inputs");
	}
}
```

Math.java

```
package com.nt.service;
public class Math {
	public float div(int x,int y){
		if(y==0)
			throw new ArithmeticException(" Divider can not Zero ");
		else
			return (float)x/y;
	}
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
```

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
<bean id="math"  class="com.nt.service.Math"/>
<bean id="exLogAdvice" class="com.nt.aspect.CommonExceptionLogger"/>
<aop:config>
<aop:pointcut  id="ptc1" expression="execution(* com.nt.service.*.*(..))"/>
<aop:aspect id="exLogAspect"  ref="exLogAdvice">
<aop:after-throwing method="exceptionLogging"
                                    pointcut-ref="ptc1" throwing="ae"/>
</aop:aspect>
</aop:config>
</beans>
```

ClientAppp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.Math;
public class ClientApp {
        public static void main(String[] args) {
                //create IOC container
                ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //GEt Proxy object
                Math proxy=ctx .getBean("math",Math.class);

                //call B.method
                try{
                System.out.println("Result="+proxy.div(10,20));
                System.out.println("----------------------------------------");
                System.out.println("Result="+proxy.div(10,0));
                }
                catch(ArithmeticException ae){
                        System.out.println("Internal Problem");
                }
        }
}
```

Annotations driven AspectJ AOP/Annotation based AspectJAOP/@AspectJ approach AOP:

Here we don't use xml file to configure advice classes and advice methods rather we use list of annotations given by aspectjAOP.

@Aspect------>to configure spring bean as AspectClass.(Should be used along with @component*

& @ Service)

@Before------> to configure method as before advice method.

@Around------->to configure method as around advice method.

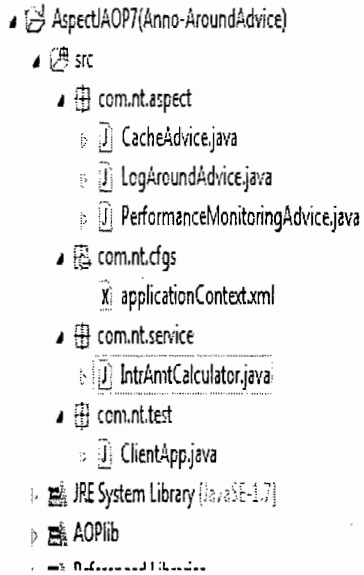
@AfterReturning----->to configure method as AfterAdvice

@AfterThrowing----->to configure method as throw advice
@Pointcut ------->to specify pointcut expression with reusability
@Order---->to specify order

- *To make IOC container recognizing all these annotations wee need to place <aop:aspectJ-autoproxy> tag in xml file.*
- While working with AspectJAOP we can take Advice classes as pojo classes and we can place multiple advice logics in a single class.

@Around, @AfterReturning, @Before, @AfterThrowing annotations allow to place pointcut expressions directly.

- To place pointcut expression with reusability we can use @Pointcut.



CacheAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import java.util.HashMap;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
@Aspect
@Order(1)(//Useful to specify Order of applying advice when they are configured through
annotations. Highvalue indicates low priority and low value indicates high priority among
 The numbers that are given )
@Component
public class CacheAdvice {
        HashMap<String,Object>  cache=new HashMap<String,Object>();

        @Around("execution(* com.nt.service.*.*(..))")
       public Object  caching(ProceedingJoinPoint pjp)throws Throwable{
              String key=pjp.getSignature()+Arrays.toString(pjp.getArgs());
```

```
            if(!cache.containsKey(key)){
                    Object retVal=pjp.proceed();
                    cache.put(key,retVal);
                    System.out.println("From taget method");
                    return retVal;
            }
            else{
                    System.out.println("From cache");
                    return cache.get(key);
            }
        }//caching
}//class
```

LogAroundAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
@Aspect
@Order(2)
@Component
public class LogAroundAdvice {
        @Around("execution(* com.nt.service.*.*(..))")
        public Object logging(ProceedingJoinPoint pjp)throws Throwable{
                System.out.println("Entering into "+pjp.getSignature()+" with
args"+Arrays.toString(pjp.getArgs()));
                //Get args and modify args
                Object args[]=pjp.getArgs();
                if((Float)args[0]<=50000){
                        args[2]=(Float)args[2]-1;
}
                Object retVal=null;
                if((Float)args[0]==0 ||(Float)args[1]==0 ||(Float)args[2]==0 )
                         throw new IllegalArgumentException("Provide +ve values as the p,t,r
values");
                else
                        retVal=pjp.proceed(args);
                // add Service tax on intr amt
                retVal=(Float)retVal+(Float)retVal*0.05f;
     System.out.println("Exiting from "+pjp.getSignature()+" with
args"+Arrays.toString(pjp.getArgs()));
     return retVal;
        }//method
}//class
```

PerformanceMonitoringAdvice.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
@Aspect
@Order(3)
@Component
public class PerformanceMonitoringAdvice {
   long start,end;
   @Around("execution(* com.nt.service.*.*(..))")
        public Object pMonitor(ProceedingJoinPoint pjp)throws Throwable{
                start=System.currentTimeMillis();
                Object retVal=pjp.proceed();
                end=System.currentTimeMillis();
                System.out.println(pjp.getSignature()+" with"
+Arrays.toString(pjp.getArgs())+" has taken "+(end-start)+"ms");
                return retVal;
                }
}
```

IntrAmtCalculator.java

```
package com.nt.service;
import org.springframework.stereotype.Component;
@Component("targetBean")
public class IntrAmtCalculator {
        public float  calcSimpleIntrAmt(float p,float t,float r){
                return (p*t*r)/100.0f;
        }
        public float calcCompoundIntrAmt(float p,float t,float r){
                return (float) (p*Math.pow((1+r/100.0f),t)-p);
        }
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<!-- <bean id="targetBean" class="com.nt.service.IntrAmtCalculator"/>
<bean id="pMAdvice" class="com.nt.aspect.PerformanceMonitoringAdvice"/>
```

```
<bean id="logAdvice" class="com.nt.aspect.LogAroundAdvice"/>
<bean id="cahceAdvice" class="com.nt.aspect.CacheAdvice"/> -->
<aop:aspectj-autoproxy/>
<context:component-scan base-package="com.nt.aspect,com.nt.service"/>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.IntrAmtCalculator;
public class ClientApp {
        public static void main(String[] args) {
                //create IOC container
                ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
                //GEt Proxy object
                IntrAmtCalculator proxy=ctx.getBean("targetBean",IntrAmtCalculator.class);
                //Call target method
        System.out.println("Simple Intr Amt"+proxy.calcSimpleIntrAmt(10000,20,2));
        System.out.println("-----------------------------------------");
        System.out.println("Compound Intr Amt"+proxy.calcCompoundIntrAmt(60000, 20, 2));
        System.out.println("-----------------------------------------");
        System.out.println("Compound Intr Amt"+proxy.calcCompoundIntrAmt(60000, 20, 2));
        }
}
```

- ⇨ While working with AspectJAOP with xml the order they follow to apply advices on target method will be decided based on the order that is used to configure advices classes as spring beans in applicationContext.xml

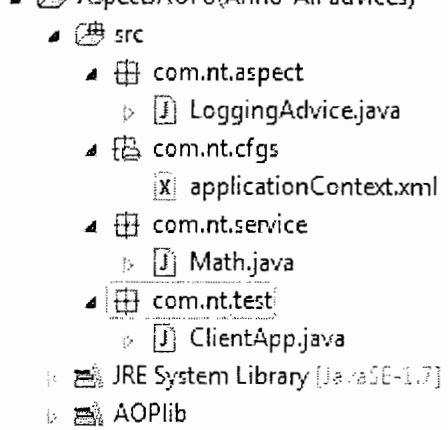
- ⇨ If advice/aspect classes are not configured in xml fie and they are configured through annotations like
  1) @Component , @Aspect then we can define the order by using
  2) @Order(-) annotations
  3) @Componet or @Service can be used to configure ordinary java class as spring bean.
  4) @Aspect can't be used to configure ordinary java class as aspect class. But it can be used to configure spring bean as aspect class.

While working with annotation based AspectJAOP
<aop:aspect-autoproxy> is alternate to <aop:config> tag to enable aspectjAOP based on the annotations.
ThrowsAdvice not only executes for the exception that is raised in target method, it also executes for the exception that is raised in advices that are related to target method.

- ⇨ When we define multiple advice methods in single aspect class having same pointcut expression then instead of defining that pointcut expression separately for every method it is recommended to take dummymethod in same Aspect class having pointcut expression defined through

@Pointcut and specify that dummy method name in every advice method as pointcut expression.

```
@Aspect
public class MyASpect{
@Pointcut("execution(*com.nt.service,*.*(..))")
public void myPointcut(){ //Dummy method representing Pointcut expression
}
@Around(value="myPointcut()")
public Object log(ProceedingJointPoint pjp){
.....
.....}
@Before(value="myPontcut()")
public void audit(JoinPoint jp){
....
....}
```



eg:

LoggingAdvice.java

```
package com.nt.aspect;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;
@Component
@Aspect
public class LoggingAdvice {
        @Pointcut("execution(* com.nt.service.Math.*(..))")
        public void myPointcut(){
                }
        @Around("myPointcut()")
        public Object  log(ProceedingJoinPoint pjp)throws Throwable{
                System.out.println("Entering into "+pjp.getSignature());
                Object retVal=pjp.proceed();
                System.out.println("Leaving from "+pjp.getSignature());
```

```
            return retVal;
        }
        @Before("myPointcut()")
        public  void checkInputs(JoinPoint jp){
            Object args[]=jp.getArgs();
            if((Integer)args[0]<=0 ||(Integer)args[1]<=0 ){
                throw new IllegalArgumentException("Provide proper inputs");
            }
        }
        @AfterReturning(value="myPointcut()",returning="result")
        public void checkResult(JoinPoint jp,int result){
            if(result<=0)
                throw new IllegalArgumentException("InValid result");

        }
        @AfterThrowing(value="myPointcut()",throwing="e")
        public void exceptionLogger(JoinPoint jp,Exception e){
            System.out.println(jp.getSignature()+" has raised"+e.getClass()+" Exception");
        }
}
```

Math.java

```
package com.nt.service;
import org.springframework.stereotype.Component;
@Component("math")
public class Math {
            public int sum(int x,int y){
            return x+y;
        }
            public int sub(int x,int y){
            return x-y;
        }
            public int mul(int x,int y){
            return x*y;
        }
        public  int div(int x,int y){
            return (int)x/y;
        }
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
```

```
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<aop:aspectj-autoproxy/>
<context:component-scan base-package="com.nt.aspect,com.nt.service"/>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.Math;
public class ClientApp {
	public static void main(String[] args) {
		//create IOC container
		ApplicationContext ctx =new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
		//GEt Proxy object
		Math proxy=ctx .getBean("math",Math.class);
		//Call target method
		System.out.println(proxy.sum(10,20));
		System.out.println("------------------------------");
		System.out.println(proxy.sub(10,30));
		System.out.println("------------------------------");
		System.out.println(proxy.mul(20,30));
		System.out.println("------------------------------");
		System.out.println(proxy.div(20,10));
	}
}
```

Note: in real time most reguraly used AspectJ AOP is annotation driven aspect AOP.

Transaction Management

The process of combining related operations into single unit and executing then by applying do everything or nothing principle called "***Transaction Management".***

1) The withdraw amount from source account, deposite amount into dest amount must happen in Transaction envinorment.
2) In employee registration employee details insertion into Database and insertion into payroll Database most happen in Transactional envinorment.
3) In online shopping website order booking and payment must happen in transactional environment.

Transaction management allows to implement ACID properties.

A-->Atomocity
C--->Consistency
I----->Isolation
D---->Durability

- ⇨ The process of combining indivisible operations into single unit is called Atomocity. Atomocity word is derived from the term called Atom.
- ⇨ Even though rules kept on Database software (like balance must not be negative) are violated during the course Transaction management , but if there is a guarrenty that no

rule will be vailoated at the end of Transaction management , then we can say Database is consistent.

- ⇨ Applying locks on Database software and allowing one user per one application at a time to manipulate database data is called isolation.
- ⇨ Brining data back to normal state by log file, backup file though it is crashed indicated durability of Database

Every Transaction contains 3 operations:-

1). Begin Transaction

.....

..... 2). //continue Transaction (Execute logics)

.....

3).Commit/Rollback Transaction

Transaction management resource must allow data commit or rollback operations.

File can't be taken as Transaction resource. Database software, message Queues can be taken as Transaction resource. Because file doesn,t support commit and roll ~~ball~~ rollback operating.

- ⇨ Based on the Transaction management applications instruction the Transaction manager connects to resource through resource manager to commit or rollback Transaction .

Based on number of resources that are involved in a transaction there are 2 types of transaction.

- ⇨ Local Transaction

Single resource will participate in transaction boundary

eg: Transfering money operation between 2 accounts of same bank.

- ⇨ Global Transaction /Distributed Transaction /XA Transaction

Multiple resources will participate in a single transaction boundary.

eg: Transfer money operation between 2 accounts of 2 different bank

- ⇨ Global transaction runs based on 2 pc protocol (2 phase commit)

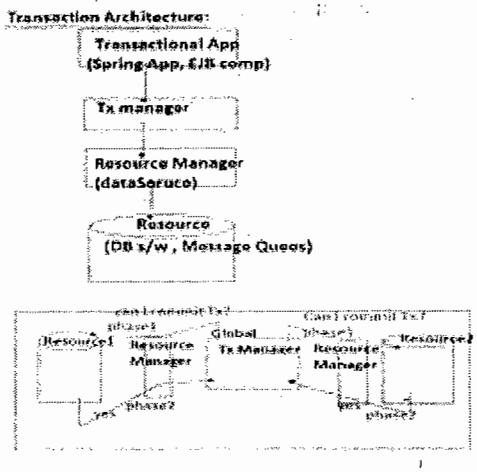
Phase-1 :

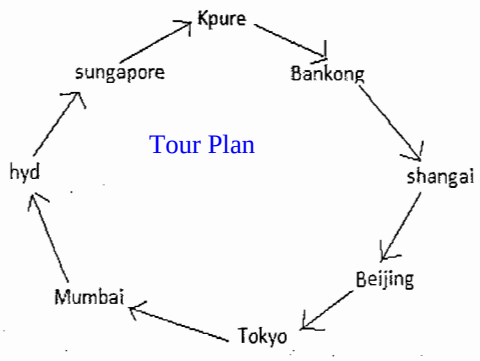
Transaction manager talks with all the resources through resource manager seeking permission to commit the Transaction by checking the data consistency.

Phase-2:

If all the resource managers respond and says "OK" then the data of multiple resources will be commited. If any resource manager says "no" then data of multiple database will be rolleback.

=>Jdbc, hibernate supports only local Transaction management . Spring, EJB supports both local and global Transaction management .To work with distributed transaction we need XA JDBC driver and XA Datasource(Like OJDBC 6-xa.jar)





Note:
Using jdbc, hibernate we can bring the effect of distributed transaction management by calling commit()/rollback() on multiple connection objects (or) session objects but that is not a good practice because if exception raises between two commit method calls the transaction management

Transaction Models:

- ⇨ When the tour plan is given to flat Transaction management model applicaton, if one journey tkts are not available it will cancel the remaining journey tkts. Because all bookings will be taken as direct operations of main transaction.
- ⇨ If above tour plan is given to nested Transaction management . Model application it will confirm remaining journery tickets even though one or another journey ticket are not available . Becase here each journery ticket booking will be taken as sub Transaction management of main Transaction management so the success or failure of one sub transaction doesn't affect other sub Transaction .

1.Flat Transaction

Main Transaction{
journey1 tkt booking
Journey2 tkt booking
.....
.....
Journeryn tkt booking
}

⇨ **2.Nested Transaction**

```
Main tx{
Sub tx1{
.....
.....
}//Journey 1 ticket booking
....
Sub tx2{
.......
.....//journey ticket booking
}
}
```

- Jdbc, Hibernate, EJB supports only flat transactions spring supports both flat Transaction management and Nested Transaction management .
- Generally we use JDBC code for local Transcation Management as shown below

Local Transaction management sample code using JDBC:

```
try {
con.setAutocommit(false);  // Transaction management   began
.....
.....                //Transaction management .logic
.....
con.commit ();
}catch (Exception e){
con.rollback();
}
```

- To perform distributed Transaction management we need either application server (JEE server) or Third party api like atomokis that gives userTransaction Object as transaction manager. We also need XA jdbc drivers, XA Datasource manager to get Transaction manager required for distributed Transaction management .

- Here we never directly call con.commit() or con.rollback. we ask TransactionManager to perform these operations by getting connection through Resource Manager (Datasource). The server like weblogic supply "userTransaction" object in its jndi registry having fixed jndi name like "javax/transaction/userTransaction".

- We use JTA support to perform global Transacation management in programmatic approach. We use EJB to perform Global Transaction management in declarative approach (xml configuration/ annotation configuration).

- In programmatic approach we write Transaction management code inside the B.methods. So disabiling Transaction management service is very complex .

- In declarative approach b.method contains only business logic but the Tx Transaction configuration will be placed in xml file. The container/server/framework enable Transaction management service on b.methods based on xml file.

Jdbc, JTA supports only programmatic Transaction management .
EJB supports both programmatic and declarative management.

<u>Global Transaction management in the component that is deployable in the server using JTA</u> (programmatic appraoch)

```
InitialContext IC=new InitialContext();
UserTransaction ut=( UserTransaction)IC.lookup ("javax/transaction/userTransaction");
try{
Ut.begin(); //begin Transaction management
.....
.....                Transaction management .logic
.....
ut.commit();
}catch (Exception e){
```

```
ut.rollback();
}
}
```

Declarative Transaction management Management Using EJB:

EJB component

```
public class Mycomp implements SessionBean
{
public void bml()
{
.....
.....      // only B.logic or no Transaction management logic
.....
}
}
```

Ejb-jar.xml

- ⇨ Bean class configuration
- ⇨ **Transaction management configures**

Note: Based on xml configuration the EJB container appiles Transaction management services on to b.methods of EJB component.

The EJB's declarative Transaction management service is really nice But it is heavy weight.

Spring Transaction:

- ⇨ In the above discussion we need to use different java, JEE apis or 3rd party apis like JPA, hibernate and etc ..... To perform different modes of Transaction management.
- ⇨ Due to this switching from local Transaction management to distributed or switching from one Transaction management implementation to another Transaction management implementation become very complex for the programmer.
- ⇨ Spring provides unified envinorment to work with any Transaction management because it is a framework providing abstraction layer on multiple technologies.
- ⇨ Due to this we can make spring to use internally any implementation of Transaction management . But we can write the unified code for Transaction management.
- ⇨ Due to this switching to distributed Transaction management from local Transaction management and switching from one Transaction management implementation to another Transaction management implementation doesn't make to change spring code much.

Spring supports 4 ways of transaction management:-

1) Programatic Transaction management
2) Declarative Transaction management using spring AOP
3) Declaraive Transaction management using AspectJAOP
4) Annotations based Transaction management using AspectJAOP
5) AspectjAOP 100% code Tx mgmt
6) AspectjAOP Spring Boot Tx mgmt(a)

- ⇨ In spring Transaction service is secondary logic/cross cutting concern. It is given as a pre-defined Aspect logic as the combination of Around advice and throwsAdvice type.

Transaction management will begin while entering into method and will be commit when method returns a value or will be rolledback when methods throws exceptions. So the Transaction management service is combination of AroundAdvice and throwsAdvice.

- In order to perform Transaction management we need to choose the appropriate Transaction manager/ Transaction management coordinator
  a) DataSourceTransactionManager---->For underlying JDBC code (Local Transaction management)
  b) HibernateTransactionManager------>for hibernate code(Local Transaction management)
  c) JDOTransactionManager------> for the JDO code (Local Transaction management)
  d) JtaTransactionManager--------->for Distributed Transaction management and etc.....

Declarative Transaction management using spring AOP:

- TransactionProxyFactoryBean----->Generates proxy class applying Transaction service on Target methods.
- NameMatchTransactionAttributeSource----->Allows to specify Transaction management attribute on target methods it is like pointcut

Note:

- To create DataSourceTransactionManager the "dataSource" object is dependent.
- In programmatic Transaction management we write Transaction management code inside the b.method which is against of AOP, so don't use programmatic Transaction management management.
- In declarative Transaction management we write Transaction management related configuration in xml file so target method just contain business logic. But Transaction management service will be applied on business method because of proxy class
- While working with any mode declarative transaction management we never begin (or) commit (or) rollback transaction explicitly. All these operations will be taken care by transaction manager with the support of proxy class the transaction manager begins the transaction while entering into the business method and commit the transaction at the end of the business method if no unchecked exception is raised. Otherwise rollbacks the transaction .

Steps to work with spring AOP declarative transaction manager

1) Configure the target class/service class.
2) Configure transaction manager as transaction advice.
3) Configure transaction ProxyFactoryBean injecting transaction manager, target, specifying transaction attribute to generate proxy class having transaction service.

- TxMgmtProj1(Decl Txmgmt Spring AOP Local ,Flat T
  - src
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.dao
      - BankDAO.java
      - BankDAOImpl.java
    - com.nt.service
      - BankService.java → BankServiceImpl.java
    - com.nt.test
      - ClientApp.java
  - Referenced Libraries
  - TXlib
    - spring-jdbc-4.1.6.RELEASE.jar - C:\Users\dell\
    - spring-tx-4.1.6.RELEASE.jar - C:\Users\dell\De
    - ojdbc14.jar - D:\oracle\product\10.2.0\db_2\
    - spring-aspects-4.1.6.RELEASE.jar - C:\Users\d
    - spring-beans-4.1.6.RELEASE.jar - C:\Users\del
    - spring-context-4.1.6.RELEASE.jar - C:\Users\d
    - spring-context-support-4.1.6.RELEASE.jar - C:
    - spring-core-4.1.6.RELEASE.jar - C:\Users\dell\
    - spring-expression-4.1.6.RELEASE.jar - C:\Users
  - JRE System Library [jre7]

BankDAO.java

```
package com.nt.dao;
public interface BankDAO {
	public int withdraw(int acno,int amt);
	public int deposit(int acno,int amt);
}
```

BankDAOImpl.java

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
public class BankDAOImpl implements BankDAO{
	private JdbcTemplate  jt;
	private static final String WITHDRAW_QRY="update  tx_account set balance=balance-?
where accno=?";
	private static final String DEPOSITE_QRY="update  tx_account set balance=balance+?
where accno=?";
	public void setJt(JdbcTemplate jt) {
		this.jt = jt;
	}
	@Override
	public int withdraw(int acno, int amt) {
		int cnt=jt.update(WITHDRAW_QRY,amt,acno);
		return cnt;
	}
	@Override
	public int deposit(int acno, int amt) {
```

```
            int cnt=jt.update(DEPOSITE_QRY,amt,acno);
            return cnt;
        }
}
```

BankService.java

```
package com.nt.service;
import com.nt.dao.BankDAO;
public class BankService {
        private BankDAO dao;
            public void setDao(BankDAO dao) {
            this.dao = dao;
        }
        public boolean transferMoney(int srcNo,int destNo,int amt){
            int result1=dao.withdraw(srcNo, amt);
            int result2=dao.deposit(destNo, amt);

            if(result1==0 || result2==0){
                throw new RuntimeException("Money not Transfered(Tx rolled back)");
            }
            else{
                System.out.println("Money Transfered(Tx committed)");
                return true;
            }
        }
}
```

applicationContext.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-2.0.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="bankDAO"  class="com.nt.dao.BankDAOImpl">
<property name="jt" ref="template"/>
</bean>
<bean id="txMgr"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager"><property
name="dataSource"  ref="dataSource"/>
</bean>
<bean id="bankService"  class="com.nt.service.BankService">
<property name="dao" ref="bankDAO"/>
```

```
</bean>
<bean id="nmtas"
class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSource">
<property name="properties">
<props>
<prop key="transferMoney">PROPAGATION_REQUIRED</prop>
</props>
</property>
</bean>
<bean id="tpfb"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="transactionManager" ref="txMgr"/>
<property name="target" ref="bankService"/>
<property name="transactionAttributeSource"  ref="nmtas"/>
</bean>
</beans>
```

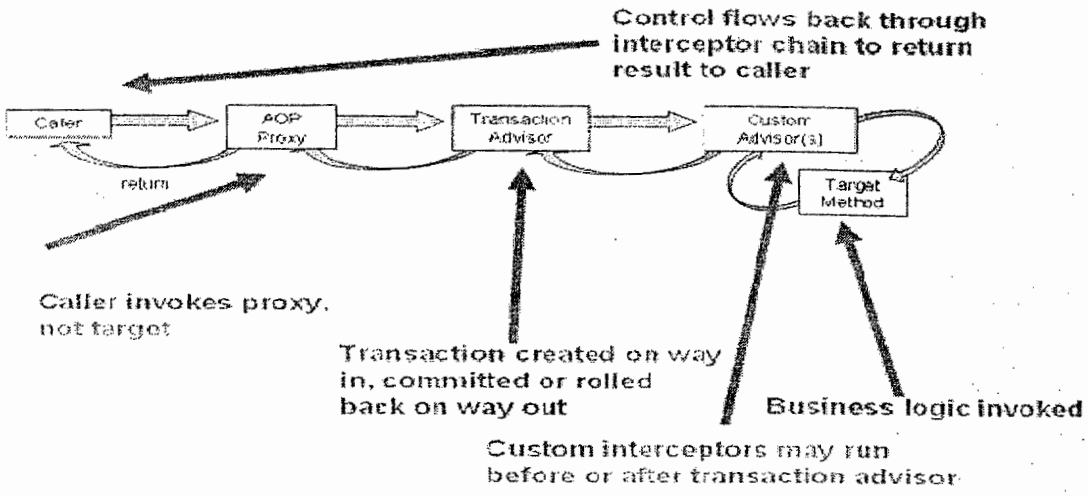
ClientApp.java

```
package com.nt.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.BankService;
public class ClientApp {
public static void main(String[] args) {
        //create IOC container
        ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //get Proxy object
        BankService  proxy=ctx.getBean("tpfb",BankService.class);
        //call b.method
        System.out.println("Money Transfered?"+proxy.transferMoney(1002, 1001,500));
}
}
```

Jar: AOPLib+springjdbcLib.

Conceptually, calling a method on a transactional proxy looks like this...

Note:- while working with declarative transaction management. the tx manager roleback the transaction only when exception is raised in bussiness method (unchecked exception). The tx manager does not rollback the transaction for checked exception.



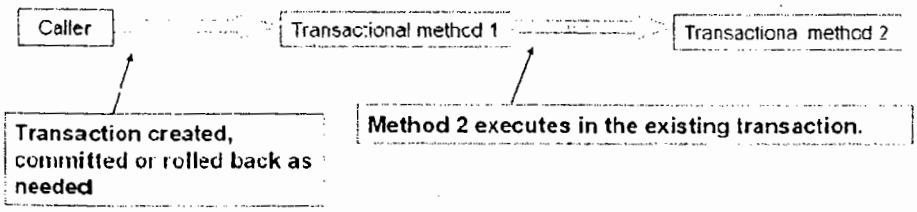
- In declarative Transaction management the Transaction management manager rollbacks the Transaction management automatically if unchecked exception is raised in the target method (RunTimeException and its subclass) otherwise the Transaction management will be commited.
- In declarative Transaction management, we can specify Transaction management attribute on the target method to make target method running with no transaction or new transaction or in client supplied Transcation.

Transaction Attribute:

Required	(PROOAGATION_REQUIRED) (//Most Regularly used)
Requirednew	(PROOAGATION_REQUIRES_NEW)
Supports	(PROOAGATION_SUPPORTS) (//default)
Never	(PROOAGATION_NEVER)
notSupported	(PROOAGATION_NOT_SUPPORTED)
mandatory	(PROOAGATION_MANDATORY)

Required:

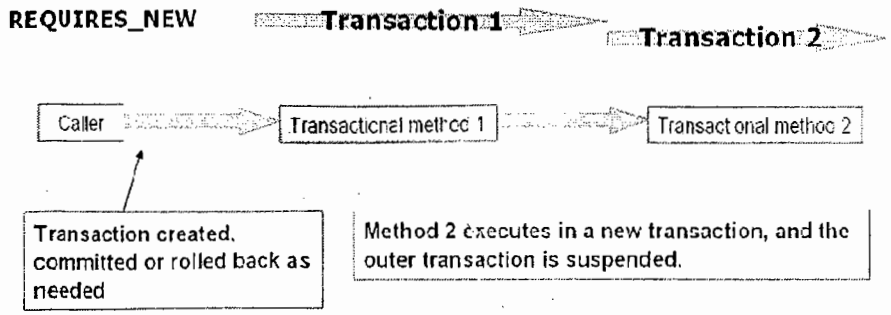
REQUIRED **Transaction**



- If clientApp calls the B.method in a Transaction management then b.method runs that Transaction management .
- If clientApp calls the B.method without Transaction management then b.method runs a new Transaction management .

Note: It is most popularly used Transaction management attribute.

RequiresNew:



⇨ B.method/target method always runs in a new Transaction management irrespective of weather clientApp calls b.method in a Transaction management or nod.

Supports:

=>B.method runs in client Application Transaction management , if client appliction callsB.methodin a Transaction management

=>B.method runs without Transaction management , if client application calls the b.method without Transaction management .

=>It is default attribute/transaction attribute.

Never:

ClientApp must not call b.method ina Transaction management , if called Exception will be raised

Not Supported:

B.method/Target method always run in No Transaction irrespective of wheather client application calls B.method with or without transaction.

Manadatory:

ClientApp must call.

B.method in a transaction otherwise Exception will be raised.

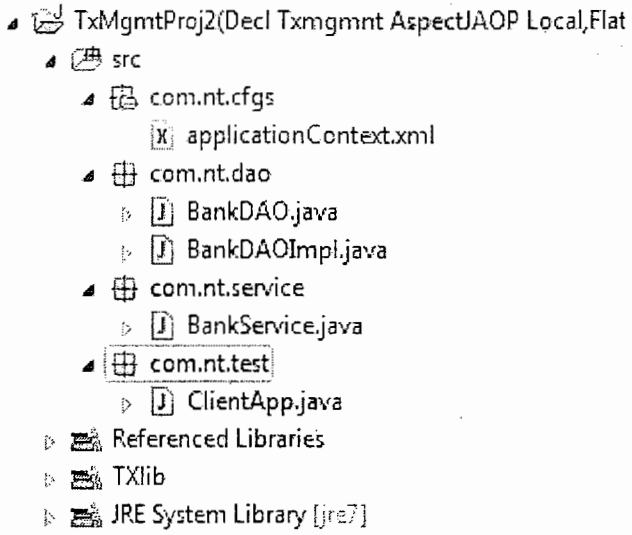
Transaction attributes table

Transaction attribute	client Transaction	Bean's Transaction
Required	none T1	(without Tx) T2 (new) T1
RequiresNew	None T1	T2 T2
Supports	None T1	None T1
Manadatory	None T1	Error T1
Not Supported	None T1	None one
Never	None T1	None error

Declarative Transaction management Using AspectJ AOP:

- It is most regular used Transaction management mechanism. To work with this we need to import AOP, tx namespacs. In order to activate Transaction management service as Advice having bean id to specify Transaction management attribute on target methods we can use <tx:advice>, <tx:attributes>, <tx:method> tags. The Transaction management advice performs commit or rollback operation through Transaction manager so we need to specify Transaction manager while configuring Transaction Advice.

- In order to apply Transaction advice on the specific methods of Target/service class use <aop:config>, <aop:advisor>, <aop:pointcut> tag as shown below.



Add AspectJ to jar files.

BankDAO.java

```
package com.nt.dao;
public interface BankDAO {
        public int withdraw(int acno,int amt);
        public int deposit(int acno,int amt);
}
```

BankDAOImpl.java

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
public class BankDAOImpl implements BankDAO{
        private JdbcTemplate  jt;
        private static final String WITHDRAW_QRY="update  tx_account set balance=balance-?
where accno=?";
        private static final String DEPOSITE_QRY="update  tx_account set balance=balance+?
where accno=?";
public void setJt(JdbcTemplate jt) {
                this.jt = jt;
        }
        @Override
        public int withdraw(int acno, int amt) {
                int cnt=jt.update(WITHDRAW_QRY,amt,acno);
                return cnt;
```

```
        }
        @Override
        public int deposit(int acno, int amt) {
                int cnt=jt.update(DEPOSITE_QRY,amt,acno);
                return cnt;
        }
}
```

BankService.java

```
package com.nt.service;
import com.nt.dao.BankDAO;
public class BankService {
        private BankDAO dao;
        public void setDao(BankDAO dao) {
                this.dao = dao;
        }
        public boolean transferMoney(int srcNo,int destNo,int amt){
                int result1=dao.withdraw(srcNo, amt);
                int result2=dao.deposit(destNo, amt);

                if(result1==0 || result2==0){
                        throw new RuntimeException("Money not Transferd(Tx rolled back)");
                }
                else{
                        System.out.println("Money Transfered(Tx committed)");
return true;
}
 }
 }
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
```

```
</bean>
<bean id="bankDAO"  class="com.nt.dao.BankDAOImpl">
<property name="jt" ref="template"/>
</bean>
<bean id="txMgr"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource"  ref="dataSource"/>
</bean>
<bean id="bankService"  class="com.nt.service.BankService">
<property name="dao" ref="bankDAO"/>
</bean>
<tx:advice id="txAdvice" transaction-manager="txMgr">
<tx:attributes>
<tx:method name="transferMoney"  propagation="REQUIRED"
                                                read-only="false"/>
</tx:attributes>
</tx:advice>
<aop:config>
<aop:pointcut id="ptc1"
                    expression="execution(*
com.nt.service.BankService.transferMoney(..))"/>
<aop:advisor advice-ref="txAdvice" pointcut-ref="ptc1"/>
</aop:config></beans>
```

ClientApp.java

```
package com.nt.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.BankService;
public class ClientApp {
public static void main(String[] args) {
        //create IOC container
        ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //get Proxy object
        BankService  proxy=ctx.getBean("bankService",BankService.class);
        try{
        //call b.method
        System.out.println("Money TransferMoney?"+proxy.transferMoney(1006, 1001,500));
        }
        catch(Exception e){
                e.printStackTrace();
        }
```

- We can repeat <tx:method> tag for multiple times to apply different Transaction management attributes on different target method. If we specify * then it represents all the methods of target class.

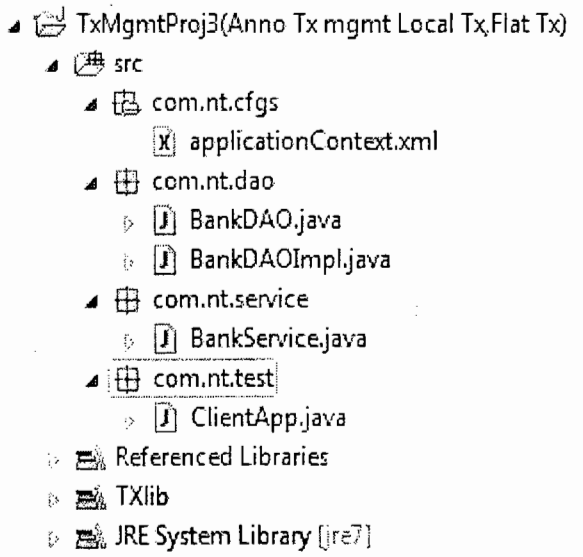
<tx:method name="*" propagation="REQUIRED"/>

- In declarative, annotation based Transaction management management the Transaction manager will rollback the transction only when target method throws unchecked exception (RuntimeException and it's subclass) doesn't rollback for checked exception.

- We Generally design DAO class one on per table statergy. Instead of applying Transaction management service on DAO class methods directly use DAO classes in service class method and apply Transaction management service on that methods of service class.

Annotations driven AspectJAOP Transaction management management:

- For this Transaction the Transaction management configurations will not be done in xml file.They will be configured directly on target class methods by using @Transactional.
- To make IOC container detecting and using these annotations we can use <tx:annotation-driven transaction-manager="...."/>



BankDAO.java

```
package com.nt.dao;
public interface BankDAO {
        public int withdraw(int acno,int amt);
        public int deposit(int acno,int amt);
}
```

BankDAOImpl.java

```
package com.nt.dao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
@Repository("bankDAO")
public class BankDAOImpl implements BankDAO{
        private JdbcTemplate  jt;
        private static final String WITHDRAW_QRY="update  tx_account set balance=balance-?
where accno=?";
        private static final String DEPOSITE_QRY="update  tx_account set balance=balance+?
where accno=?";
```

```
 @Autowired
        public void setJt(JdbcTemplate jt) {
                this.jt = jt;
        }
        @Override
        public int withdraw(int acno, int amt) {
                int cnt=jt.update(WITHDRAW_QRY,amt,acno);
                return cnt;
        }
        @Override
        public int deposit(int acno, int amt) {
                int cnt=jt.update(DEPOSITE_QRY,amt,acno);
                return cnt;
        }
}
```

BankService.java

```
package com.nt.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import com.nt.dao.BankDAO;
@Component("bankService")
public class BankService {
        private BankDAO dao;
        @Autowired
        public void setDao(BankDAO dao) {
                this.dao = dao;
        }
        @Transactional(propagation=Propagation.REQUIRED)
        public boolean transferMoney(int srcNo,int destNo,int amt){
                int result1=dao.withdraw(srcNo, amt);
                int result2=dao.deposit(destNo, amt);
                if(result1==0 || result2==0){
                        throw new RuntimeException("Money not Transfered(Tx rolled back)");
                }
                else{
                        System.out.println("Money Transfered (Tx committed)");
                        return true;
                }
        }
}
```

ApplicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
```

```
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="txMgr"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource"  ref="dataSource"/>
</bean>
<context:component-scan base-package="com.nt.service,com.nt.dao"/>
<tx:annotation-driven   transaction-manager="txMgr"/>
</beans>
```

ClientApp.java

```
package com.nt.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.BankService;
public class ClientApp {
public static void main(String[] args) {
        //create IOC container
        ApplicationContext ctx=new FileSystemXmlApplicationContext ("src/com/ nt/cfgs/
                                                        applicationContext.xml");
        //get Proxy object
        BankService  proxy=ctx.getBean("bankService",BankService.class);
        try{
        //call b.method
        System.out.println("MoneyTransfered?"+proxy.transferMoney(1001, 1008,500));
        }
        catch(Exception e){
                e.printStackTrace();
        }
}
}
```

Converting AnnotationDriven TransactionManagement application to 100% code based application

a) Remove applicationContext.xml
b) Develop applicationConfiguration class (like AppConfig.java)

```
@configuration
@ComponentScan(basePackages={"com.nt.dao", "com.nt.service"})
@EnableTransactionManagement
public class AppConfig{
@Bean(name="dbcpds")
public DataSource createDataSource{

BasicDataSource ds= new BasicDataSource();
ds.setDriverCalssName("oracle.jdbc.driver.OracleDriver");
=====
=====
        return ds;
 }
@Bean(name="jt")
public JdbcTemplate createJdbcTemplate{
JdbcTemplate jt=null;
Jt=new JdbcTemplate();
jt.setDataSource(createDataSource());
        return jt;
 }
@Bean(name="dsTxMgmr")
public FlatformTransactionManager createTxManager{
DataSourceTransactionManager txMgr =null;
Jt=new DataSourceTransactionManager ();
jt.setDataSource(createDataSource());
        return txMgr;
 }
}//class
```

c) Add @Transaction on the top of target method of target class.

```
@Transaction(Propagation=Propagation.REQUIRED,transactionManager="dsTxMgmr")
public Boolean transformoney(-,-,-){
==
}
```

d) Use AnnotationConfigApplicationContext in client application.

```
ApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
```

@Transactional properties

Property	Type	Description
value	String	Optional qualifier specifying the transaction manager to be used.
propagation	enum: Propagation	Optional propagation setting.

Property	Type	Description
Isolation	enum: Isolation	Optional isolation level.
readOnly	Boolean	Read/write vs. read-only transaction
Timeout	int (in seconds granularity)	Transaction timeout.
rollbackFor	Array of Class objects, which must be derived from Throwable.	Optional array of exception classes that**must** cause rollback.
rollbackForClassname	Array of class names. Classes must be derived from Throwable.	Optional array of names of exception classes that **must** cause rollback.
noRollbackFor	Array of Class objects, which must be derived from Throwable.	Optional array of exception classes that**must not** cause rollback.
noRollbackForClassname	Array of String class names, which must be derived from Throwable.	Optional array of names of exception classes that **must not** cause rollback.

- While working with @Transactional we can specify read only param and propagation param for transaction attributes.

- In order to apply same Transaction management attribute on all methods of target class we can place @Transactional on top of class.

Other use cases of transactionmanagement:
In employee region its details should be saved in hrDB and in finance Database.
In student register its details should be save in library database and admission Database.
In online shopping goods delivery and payment must be executed in a transaction.

```
<bean id="atomikostx" class="com.atomikos.icatch.jta.userTransaction"/>
<bean id="atomikosUT" class="com.atomikos.location.jta.UserTransactionManager">
<property name="tx" ref="atomikostx"/>
<property name="userTransaction " ref="atomikosUT"/>
</bean>
```

Distributed Transaction management:
Flat

- It uses 2pc principle,by involving multiple resources in a single Tx boundary.
- For this we need to use XA jdbc drivers, XA dara Sources and XA databases.
- Oracle all versions, mysql 5.x (or) XA database.
- To perform distributed Tx management , we use JTA Tx manager by supplying distributed TX manager Object and TXService Object.

- We can get this 2 object either from jndi registry of application server or by using third party api like –"atomikos"

Sample code:(atomikos)

Distributed TX Management:

```
<bean id="atomikos TX Mgmr" class="com.atomikos.icatch.jta.UserTransactionManager"/>
<bean id="atomikosUT"class="com.atomikos.icatch.jta.UserTransactionImp"/>
<bean id="dtXmgr"class="org.sf.transaction.jta.JtaTransactionManager">
<property name="transactionManager" ref="atomikosTxMgmr"/>
<property name="userTransaction" ref="atomikosUT"/>
</bean>
```

For web logic:

```
<bean id="jofb1" class="org.springframework.jndi.JndiObjectFactoryBean"/> //tx service
<property name="jndiName" value="javax/transaction/UserTransaction"/>
</bean>
<bean id="jofb2" class="org.springframework.jndi.JndiObjectFactoryBean"/> //tx manager
<property name="jndiName" value="javax/transaction/TransactionManager"/>
</bean>
<bean id="dtxMgmr" class="pkg.JTATransactionManager"/> //tx service
<property name="UserTransaction" ref="jofb1"/>
<property name="TransactionManager" ref="jofb2"/>
</bean>
```

eg:

- TxMgmtProj4(Anno-Distributed Tx)
  - src
    - com.nt.cfgs
      - applicationContext.xml
    - com.nt.dao
      - DepositeDAO.java
      - DepositeDAOImpl.java
      - WithdrawDAO.java
      - WithdrawDAOImpl.java
    - com.nt.service
      - AccountService.java
    - com.nt.test
      - ClientApp.java
  - Referenced Libraries
  - TXlib
  - JRE System Library [jre7]

DepositeDAO.java

```
package com.nt.dao;
public interface DepositeDAO {
  public boolean deposite(int accNo,int amt);
}
```

DepositeDAOImpl.java

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
public class DepositeDAOImpl implements DepositeDAO {
        private static final String DEPOSITE_QRY="update tx_account set balance=balance+?
where accno=?";
```

```
        private JdbcTemplate jt;

  public void setJt(JdbcTemplate jt){
        this.jt=jt;
  }
@Override
        public boolean deposite(int accNo, int amt) {
                int cnt=jt.update(DEPOSITE_QRY,amt,accNo);
                if(cnt<=0)
                 return false;
                else
                return true;
        }
}
```

WithdrawDAO.java

```
package com.nt.dao;
public interface WithdrawDAO {
  public boolean withdraw(int accNo,int amt);
}
```

WithdrawDAOImpl.java

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
public class WithdrawDAOImpl implements WithdrawDAO {
        private static final String WITHDRAW_QRY="update tx_account set balance=balance-?
where accno=?";
        private JdbcTemplate jt;
    public void setJt(JdbcTemplate jt){
        this.jt=jt;
  }
        @Override
        public boolean withdraw(int accNo, int amt) {
                int cnt=jt.update(WITHDRAW_QRY,amt,accNo);
                if(cnt<=0)
                 return false;
                else
                return true;
        }
}
```

AccountService.java

```
package com.nt.service;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import com.nt.dao.DepositeDAO;
import com.nt.dao.WithdrawDAO;
public class AccountService {
        private  DepositeDAO depositeDAO;
        private  WithdrawDAO withdrawDAO;
```

```
        public void setDepositeDAO(DepositeDAO depositeDAO) {
                this.depositeDAO = depositeDAO;
        }
        public void setWithdrawDAO(WithdrawDAO withdrawDAO) {
                this.withdrawDAO = withdrawDAO;
        }
        @Transactional(propagation=Propagation.REQUIRED)
        public boolean transferMoney(int srcAcno,int destAcno,int amt){
                boolean result1=withdrawDAO.withdraw(srcAcno, amt);
                boolean result2=depositeDAO.deposite(destAcno, amt);
                if(result1==false || result2==false){
                        throw new RuntimeException(" Money not transfered");
                }
                return true;
        }//transferMoney(-,-)
}//class
```

appliactionContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean id="orads" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init"
destroy-method="close">
<property name="uniqueResourceName" value="XAOra"/>
<property name="xaDataSourceClassName"
value="oracle.jdbc.xa.client.OracleXADataSource"/>
<property name="xaProperties">
<props>
<prop key="databaseName">xe</prop>
prop key="user">scott</prop>
prop key="password">tiger</prop>
prop key="URL">jdbc:oracle:thin:@localhost:1521:xe</prop>
/props>
/property>
roperty name="poolSize" value="10"/>
bean>
ean id="mysqlds" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init"
troy-method="close">
roperty name="uniqueResourceName" value="XAmysql"/>
```

```
<property name="xaDataSourceClassName"
value="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"/>
<property name="xaProperties">
<props>
<prop key="databaseName">ntsp7</prop>
<prop key="user">root</prop>
<prop key="password">root</prop>
<prop key="URL">jdbc:mysql:///ntsp7</prop>
</props>
</property>
<property name="poolSize" value="10"/>
</bean>
<bean id="template1" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="orads"/>
</bean>
<bean id="template2" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="mysqlds"/>
</bean>
<bean id="wDao" class="com.nt.dao.WithdrawDAOImpl">
<property name="jt"  ref="template1"/>
</bean>
<bean id="dDao" class="com.nt.dao.DepositeDAOImpl">
<property name="jt"  ref="template2"/>
</bean>
<bean id="service" class="com.nt.service.AccountService">
<property name="depositeDAO" ref="dDao"/>
<property name="withdrawDAO" ref="wDao"/>
</bean>
<!-- Construct Atomikos UserTransactionManager, needed to configure Spring -->
<bean id="atomikosTxMgmr"  class="com.atomikos.icatch.jta.UserTransactionManager"
    init-method="init" destroy-method="close">
<!-- when close is called, should we force transactions to terminate or not? -->
<property name="forceShutdown" value="true"/>
</bean>
<!-- Also use Atomikos UserTransactionImp, needed to configure Spring -->
<bean id="atomikosUT" class="com.atomikos.icatch.jta.UserTransactionImp">
<property name="transactionTimeout" value="300"/>
</bean>
<!-- Configure the Spring framework to use JTA transactions from Atomikos -->
<bean id="dtxmgr" class="org.springframework.transaction.jta.JtaTransactionManager">
<property name="transactionManager" ref="atomikosTxMgmr"/>
<property name="userTransaction" ref="atomikosUT"/>
</bean>
<tx:annotation-driven   transaction-manager="dtxmgr"/>
</beans>
```

ClientApp.java

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import com.nt.service.AccountService;
public class ClientApp {
public static void main(String[] args) {
        //create IOC container
        ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //get Proxy object
        AccountService  proxy=ctx.getBean("service",AccountService.class);
        try{
        //call b.method
        System.out.println("Money Transfered?"+proxy.transferMoney(1201, 1002,500));
        }
        catch(Exception e){
                e.printStackTrace();
        }
}
}
```

Jars: Spring jdbclib, AOP library, Aspectjrt.jar, AspectJweaver.jar

Atomikos-transaction-api.jar, atomikos-transactions-jta.jar, atomikos-util.3.7.0.jar, com.mysql.jdbc-5.1.5.jar, jta.jar ojdbc.jar, transactions-3.7.0.jar, Transactions-jdbc-3.7.0.jar.

Database table in Oracle:

Tx_account

Accno	**holder**	**balance**
101	Raja	9000

Database table in mysql 5.x (logical database:ntsp 27)

Tx _account

Accno	**holder**	**balance**
102	Ravi	8000

- ⇨ While working with distributed Transaction management the UserTransaction object represent Transaction service and TransactionManager object represents the Transaction manager that performs commit or rollback by using Transaction service.
- ⇨ JtaTransactionManager can use either server supplied or 3rd party AP supplied Tx manager, UserTransaction objects to perform Tx management.

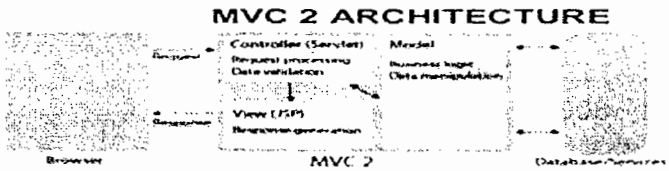
Spring MVC

Different web app development

Model1	**Model2**
Here either jsps or Servlets will be taken as main web resource programs having multiple logics.	In model 2 we take multiple technologies in multiple layers to develop various logic. This gives clean separation between logics.
Since every servlet/jsp containsmultiple logics, there is no separation between logics.So this architecture is not suitable for large scale web application.	In MVC2 architecture we take separate resources for view,separate resources for model and separate resource for controller
MVC1 architecture we take single resource (server/jsp) having view, controller logics and we take separate resources for model layer logics.	

M---->**Model**---->Data+B.logic+persistence logic
V---->**View**------>persistence logic
C----->**controller**--->integration logic/controller logic.
Controller maintains logic all the activities of web application like takes the request, passes the request to appropriate model component, gathers requests from model and passes the results to view.

MVC2 Architecture:



* **MVC 2 isolates the presentation logic from business logic.**
* **In MVC 2 controller is servlet and model is java class.**
* **Supports re usability of application components.**

Diagram:

Multiple backend software together is called EIS.
The project that is developed by using old technology or old versions of existing technologies is called Legacy project.

eg: COBOL project, Java 1.0 porject

Advantages of MVC2:

Because of multiple layer we can get clean separations between logics.
The modification done in layer logics does not affect other layer logics.
Maintainace and enhancement, so gives good productivity.
While using EJB, spring in the model layer we can take advantage of built-in middleware services.

Disadvantages:

Knowledge on more technologies are required.
For parallel development more programmers are required.
Every web application contains some common logic to handle along with application logics.

⇨ **Request graphing/wrapping:**
Reading data from the form comps of form pages and placing that data in a java class object.

⇨ **Request Submission/form submission:**
We get request from different places of web pages like through hyperlinks, through submit buttons, through other components.
These request may come with or without data, so we should handle all the request properly.

⇨ **Form Validation/Form handling:**
Verifying the pattern, format of form data and displaying same form page with errors having old values in form component.

If we develop web application by using JEE technologies directly we need to write the above **common logics** in multiple servlet programs, which leads boiler plate code problem.

These are like system services(security, logging and etc...).

If we develop MVC2 architecture web applications by using JEE technologies directly (Servlet, JSP) we need to develop all the logics of all the layers manually moreover we need to write both common and application specific logics.

MVC2 Design pattern/Architecture rules:

a) Every layer is designed to have specific logics, so place only those logics don't place any additional logics.
b) All the operations of application must be take place under the control or monitoring of controller servlet.
c) There can be multiple resources in view layer, multiple resources in model layer but there must be only 1 servlet acting as controller. This must be taken as front controller.
d) The view and model layer comps must not interact with each other directly. They must intract through controller.

If we develop mvc2 architecture based web application manually. All the logics of all the layers (including common, application specific logic) must be taken care by programmer.

If we develop mvc2 architecture based web application by using web framework then the common logics will be taken care by the framework by giving one built-in servlet as frontController so the programmer just need to develop application specific logics.

The common logics are like request wrapping, form/request submission, form handling services.

The web frameworks are:-

- **Structs** ----------> apache
- **JSF**----------->from SunMS (Oracle) **(2)**
- **Webwork**----------> from opon symphony
- **ADF** ----------->from Oracle corp **(3)**
- **SpringMVC**----------> from inteface21 **(1)**

Q) While developing a MVC2 architecture based web application can we change the roles of various technologies that are used?
Ans) Yes, we can change but it is not recommended.

Servlet as View

Presentation logic changes at regular intervals during modification in JSP is quite easy compare to doing modiication in servlet components. So it is rercommended to use jsp as view...

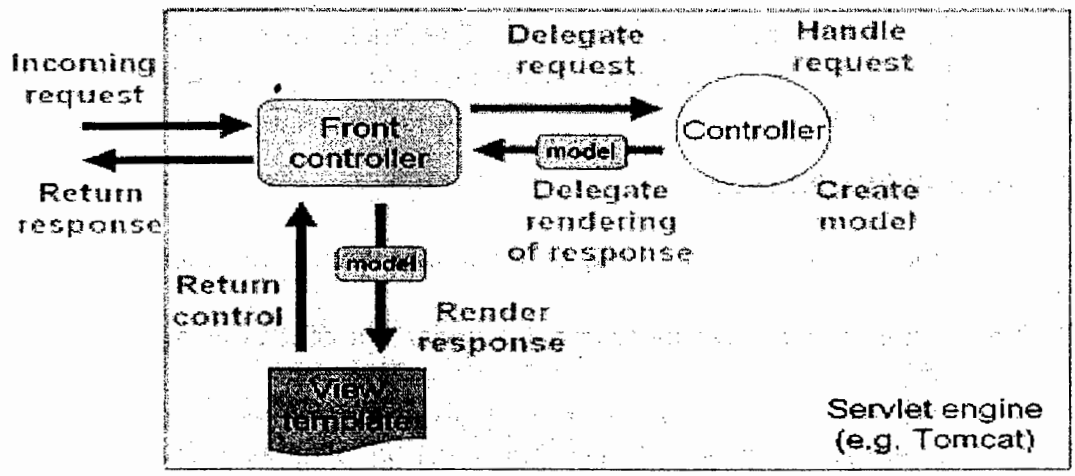
JSP as Controller

Writing Java code in JSP is a bad practice when we use JSP as a controller we should place JAVA code in JSP to interact with model layer business component.

So it is recommended to use servlet /Servlet filter as controller where placing java code is not a problem.

Java class /EJB component/Spring bean as View and controller components
This thought is wrong because they cannot take requests from clients and they cannot deliver response to browser.

Front ControllerDiagram:



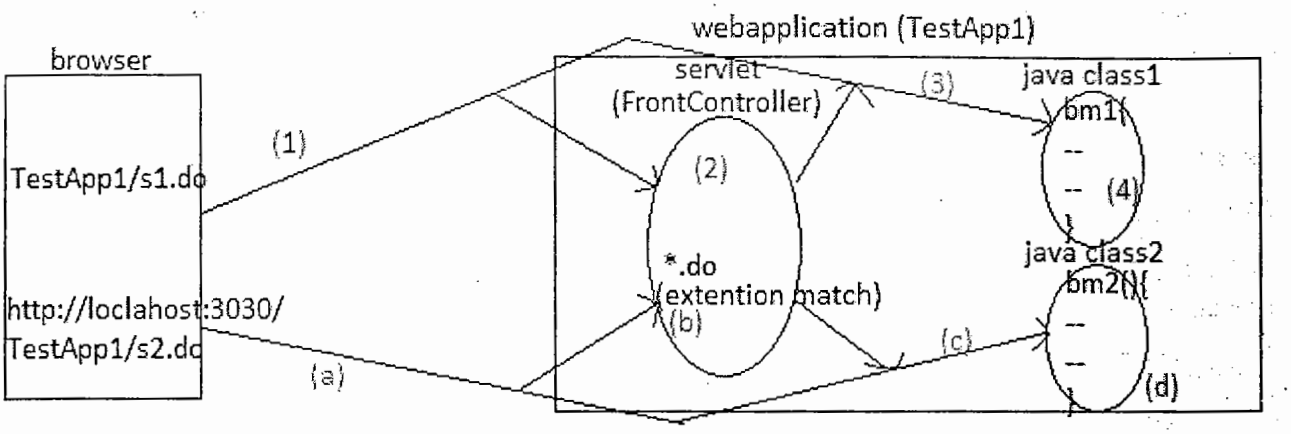
- The special web component of web application that acts as entry point for all the request and applies all common system services for all the request like security, logging and etc is called front controller.
- We can take either servlet/JSP for front controller. To make front controller traping and taking multiple request coming from clients we must configure it either with directory match or extension match url pattern.

=> *do,*.htm,*.http and etc are extension match url patterns.
=> /x/*, /x/y/* and etc are directory match url pattern.

In struts **ActionServlet** is built-in front controller servlet.

In spring MVC **DispatcherServlet** is built-in front controller servlet.

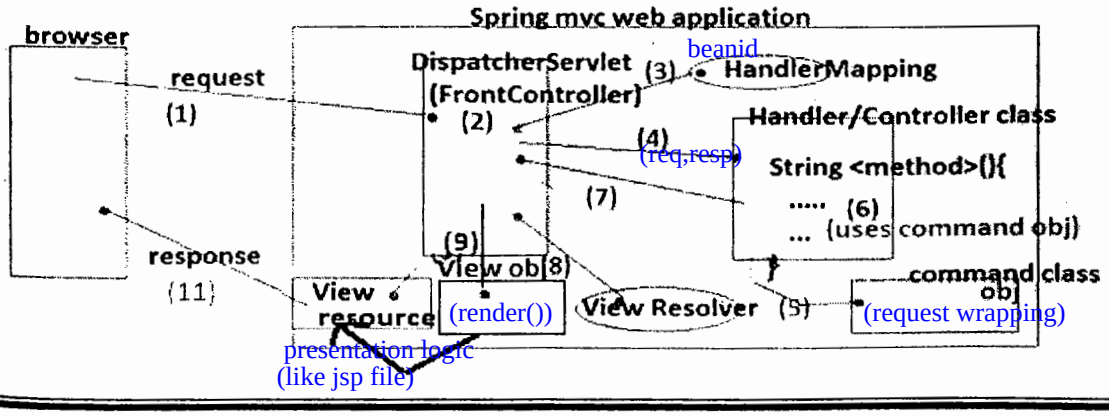
⇨ Java classes of web application cannot take request directly from browser. So we make those request getting traped by front controller to deligates the request to appropriate Java class...



- We take ServletFilter as InterceptingFilter having optional servicers to apply on web app having the ability to enable or disable (like allowing only certain browser request and etc...).
- Use FrontController to apply common system services on all the request (like security, logging and etc...).
- Use InterceptingFilter to apply optional servicers having the ability to enable or disable.
- Don't configure FrontController servlet with /* url pattern to allow certain request directly going to web comps without applying system services.
- If needed we can place both FrontControllers, InterceptingFilters in one web application development.

Note: Command class is a java Bean Whose obj holds received form data. Handler/Controller class creates Command class obj and performs request wrapping on it.

Spring MVC web application Flow



(1)Browser gives request to web application

(2)As frontcontroller Dispatcherservlet traps and takes the request and applies the common system services like security and etc....

(3)DispatchServlet uses HandlerMapping component to decide the Handler class to utilize based on the incoming request uri.

(4)Dispatcherservlet passes the control to Handler class by calling method.

(5)Handler class internally writes the received form data to command class object.

(6)Handler class process the request and generates the output, if needed it also uses command object work with form data.Sometimes handler class deligates the request to other external components like ejb,webservices and etc...

(7)Handler class returns logical view name back to DispatcherServlet.

(8)DispatcherServlet uses view Reslover to get view object having view layer technology and resource name.

(9)DispatcherServlet uses the view object to pass the control to view resource.

(10& 11)The view resource format the results and sends the ouput to browser (reponse to browser).

Spring MVC features:

- ⇨ Allows to use various technologies in the view layer like html, jsp, velocity, freemarker, xslt, etc...
- ⇨ Allows to design command class (whose object holds form data) as pojo class (Need not to extend or implement spring api class/interfaces)
- ⇨ Allow to take any type properties in command class to hold the form data.
- ⇨ Supports internationalization (il8n)
  (making our app working for different Locales).
- ⇨ Allows to configure themes for rendering better presentation.
- ⇨ Allows to develop business logics, form validation logics and other logics as reusable logics.
- ⇨ Provides clean separation between logics/comps by involving different handler mapping, views, viewresolver, command classes, handler/controller classes and etc...
- ⇨ Allows to integrate with other modules of spring framework and etc..

Spring MVC is designed around **DispatcherServlet** because various operations on the request will be done by DispatcherServlet by using different components to process the Request and to deliver the response.

Even though it is predefined servlet its configuration in web.xml is mandatory. We configure this servlet with either extension match or directory match url pattern to make that servlet as frontcontroller to take multiple requests.

This servlet internally activates IOC container of type WebApplicationContext by taking <DispatcherServlet logical name in web.xml>-servlet.xml or specified filename (as namespace init param value) in spring bean configuration file.

Spring bean configure file contains Handler mapping,Controller class,viewresource and etc...

web.xml

```
<web-app>
<servlet>
    <servley-name>dispatcher </servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <int-param>
    <param-name>namespace </param-name>
```

```
    <param-value>applicationContext</param-value>
    </init-param>
    <load-on-startup>2 </load-on-startup>
    </servlet>
    <servlet-mapping>
    <servlet-name>dispatcher </servlet-name>
    <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    </web-app>
```

If this namespace init parameter is not configured the IOC container created by DispatchServlet (ds) takes dispatcher-servlet.xml as springbean configure file. If it is configured the IOC container takes applicationContext.xml as spring bean configuration file.

It is recommended to place jsp files inside WEB-INF folder or in its sub folders. The reasons are:-

1) We can hide the jsp technology that is used in the webapplication development from enduser.(protection from hackers)

2) If jsp page is displaying request attributes given by serlvet the direct request to jsp will display ugly values. We can avoid this ugly messages by not allowing direct request to jsp for that place jsp inside the WEB-INF folder.

3) If session is not logged out properly hackers can access previous session by appending sessionid to the request url of jsp collected from browser history. This can be avoided by keeping jsp's inside the WEB-INF folder and also tacking the support of filter.

4) When jsp's are place inside WEB-INF folder the underlying server/container and other web components of web application access it directly even though it is not configured in web.xml having URL pattern. Spring MVC provides to locate jsp components of WEB-INF folder for other components of application.

Controller class:-

In order to develop our controller/handler class we must take a class implementing **org.springframework.web.servlet.mvc.Controller**Interface and should provide implementation for handleRequest (-,-) method having logic to process the request and to return ModelAndView object having logical view name.

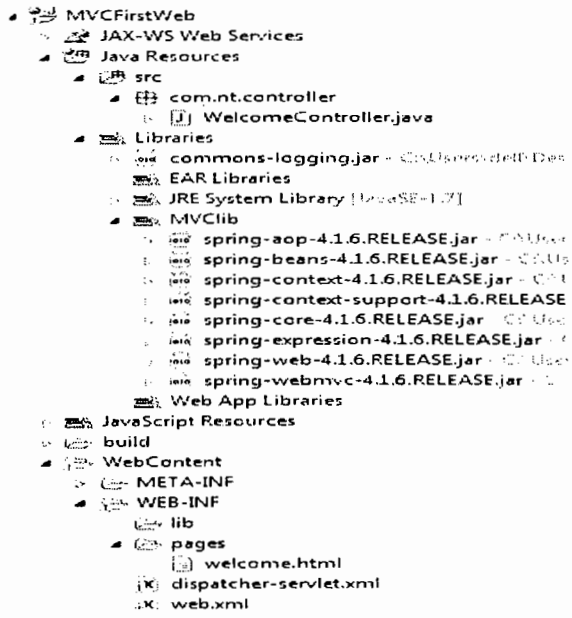
eg:

```
public class MyController implements Controller{
public ModeAndView handleRequest (HttpServletRequest req, HttpServletResponse res)
{
.............................
.............................
return ModelAndView ("home");
}
}
```

Every controller/handler class must be configure in spring configuration file having bean id and that bean id must be mapped with incoming request url by using one or other HandlerMapping.HandlerMapping is responsible to map incoming request urls to handler classes (controller classes).

Every HandlerMapping is a class that implement org.springframework.web.servlet.HandlerMapping interface the most regularly used handlermapping class is "**SimpleUrlHandlerMapping**".

```
<!controller class.....>
<bean id="wc" class="com.nt.controller.WelcomeController"/>
<!----HandleMapping>
<bean id="surl" class="org.springframework.web.servlet.SimpleUrlHanderMapping">
<property name="mappings">
<props>
<prop Key="home.htm">wc</prop>
<props>
</property>
</bean>
```



Request URL:

Request : **http://localhost:3030/MVCFirstWeb/home.htm**

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean id="wc"  class="com.nt.controller.WelcomeController"/>
<bean id="surl"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="home.htm">wc</prop>
</props>
</property>
</bean>
<bean id="vr"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

welcome.jsp:

```
<h1><center> HOME PAGE</center></h1>
${requestScope.msg}
```

WelcomeController.java:

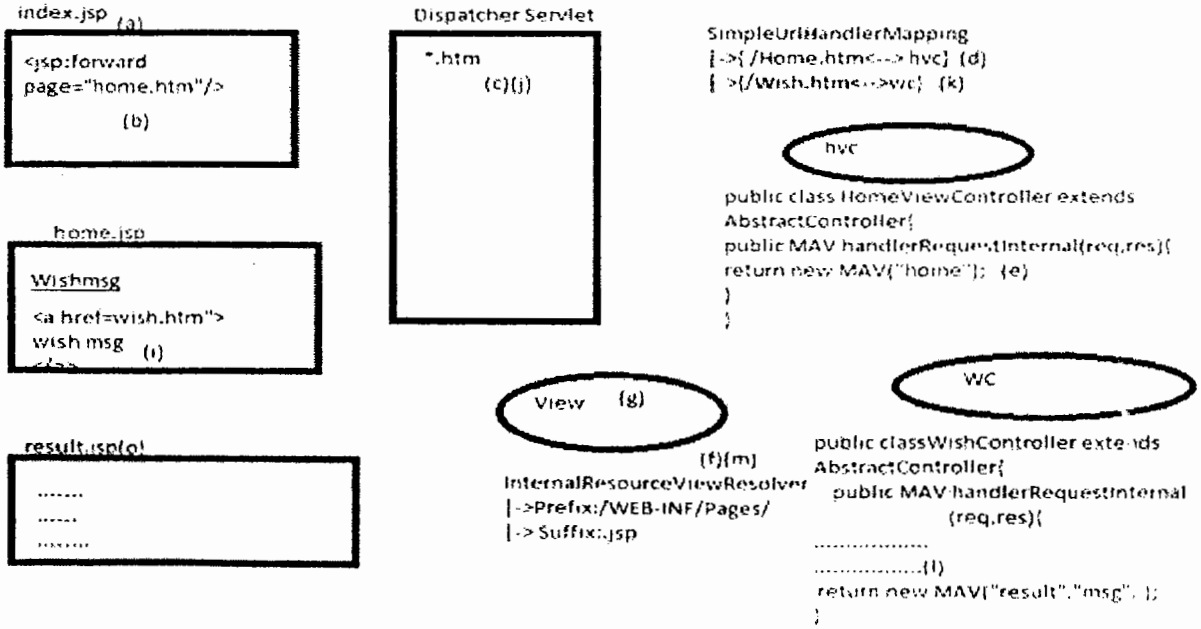
```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
public class WelcomeController implements Controller {
    @Override
    public ModelAndView handleRequest(HttpServletRequest req,
```

```
HttpServletResponse res) throws Exception {
            return new ModelAndView("welcome","msg","Welcome To Spring MVC");
    }  }
```

Viewresolver:

- This view Resolver takes the logical view name given by controller class and resolves physical view name by adding prefixes, suffixes and returns View interface implementation class object back to DispatcherServlet having physical view name.
- DispatcherServlet calls render() method on the given View implementation class object and this render() method passes the control to physical view resource.
- Every ViewResolver is a class that implements org.springframework.web.servlet.ViewResolver interface class. Spring has given multiple built-in ViewResolver classes in that InternalResourceViewResolver is useful to return view implementation class object having servlet, jsp programs placed in the WEB-INF folder of the web application (internal resource).

Spring MVC web app2 that launches homepage and process the request based on the hyperlink clicked in homepage.



Note:

- In spring mvc application only welcome pages will be placed outside the WEB-INF Folder and remaining jsps are recommended to place inside WEB-INF folder for security reason. Welcome page executes automatically for every request given to web application and gives implicit request by using <jsp:forward>tag or response.sendRedirect(-) method to launch other pages that are there inside WEB-INF folder through controller/handler classes.

- It is always recommended to develop our controller/handler class by extending from predefined XxxController classes. That means it is not recommended to developer same class by implementing Controller interface directly.

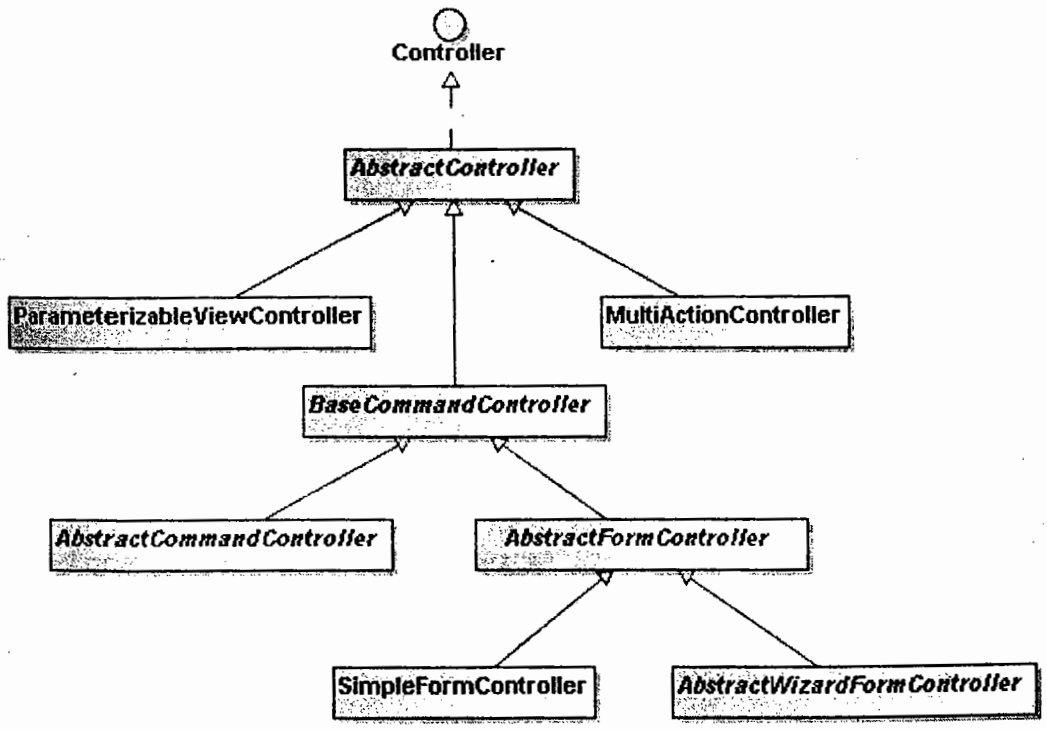
Controller/Handler:

- The java class that implements org.springframework.servlet.mvc.Controller(I) is called controller/Handler. It is mainly useful to render logical viewname and data that is required to display in view layer resource.

- Placing business logic/request processing logic in controller class is not recommended process. It is recommended to place such logics in service classes and communicate with them from controller class.

- Spring API has supplied lots of built-in controller classes implementing "Controller" interface. These classes are given to handle the request coming from form pages, hyperlinks and from other form components...

The Controller Classes are:

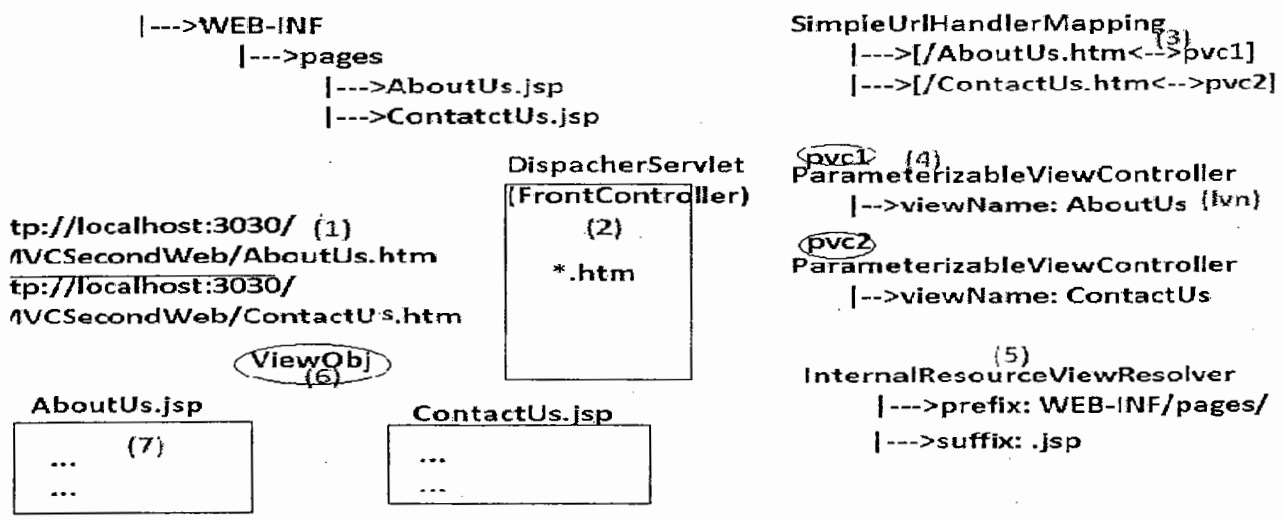


ParameterizableViewController:

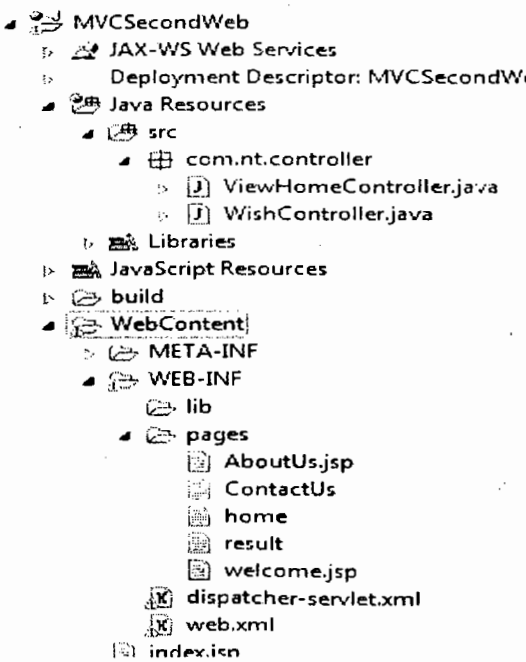
- In order to display webpage without processing request, then instead of taking separate user defined controller class we can configure this class by specifying the destination view name using "viewName" property value. This is useful to display webpage for hyperlink generated request without having any request processing logics

⇨ eg: clicking "AboutUs", "ContactUs" hyperlinks. For multiple request of this kind we need to configure this class for multiple times.

Diagram:




```
SimpleUrlHandlerMapping
        |--->[/Aboutus.htm<->pvc1]
        |--->[/contact.htm<->pvc2]
PVC1
ParameterizableViewController
 |------->ViewName: AboutUs  (logical viewname)
 PVC2
ParameterizableViewController
                |----->ViewName:ContactUs
InternalResourceViewResolver
        |------>Prefix:/WEB-INF/Pages/
        |------->Softfix:.jsp
```



Note:

- In spring mvc application only welcome pages will be placed outside the WEB-INF Folder and remaining jsps are recommended to place inside WEB-INF folder for security reason. Welcome page executes automatically for every request given to web application and gives implicit request by using <jsp:forward>tag or response.sendRedirect(-) method to launch other pages that are there inside WEB-INF folder through controller/handler classes.

- It is always recommended to develop our controller/handler class by extending from predefined XxxController classes. That means it is not recommended to developer same class by implementing Controller interface directly.

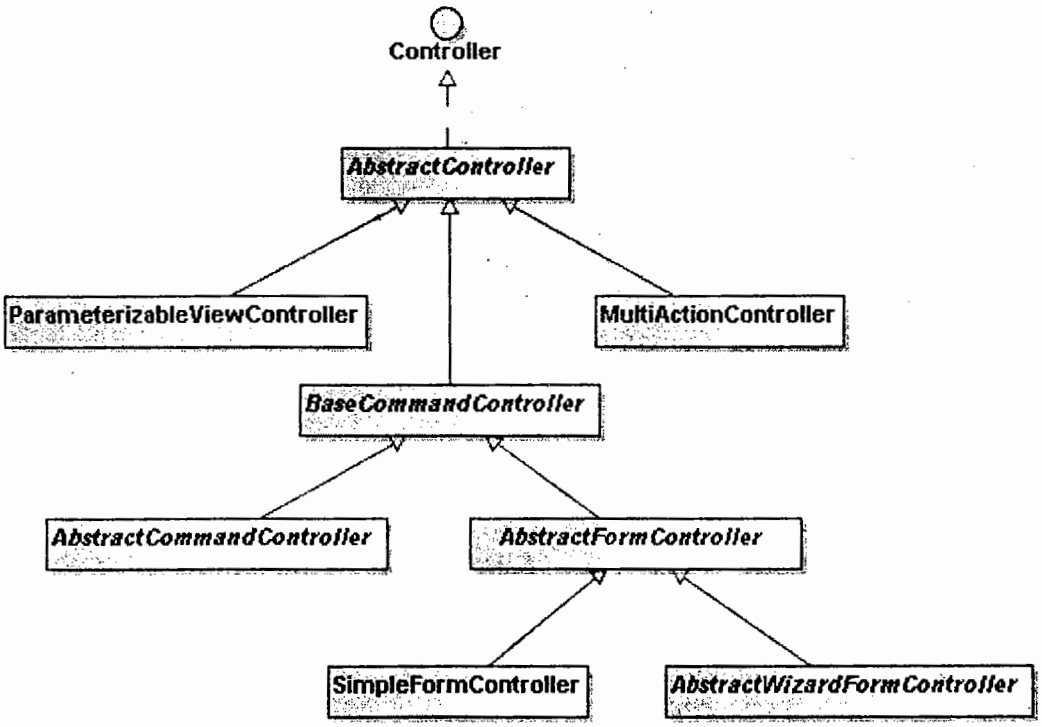
Controller/Handler:

- The java class that implements org.springframework.servlet.mvc.Controller(I) is called controller/Handler. It is mainly useful to render logical viewname and data that is required to display in view layer resource.

- Placing business logic/request processing logic in controller class is not recommended process. It is recommended to place such logics in service classes and communicate with them from controller class.

- Spring API has supplied lots of built-in controller classes implementing "Controller" interface. These classes are given to handle the request coming from form pages, hyperlinks and from other form components...

The Controller Classes are:

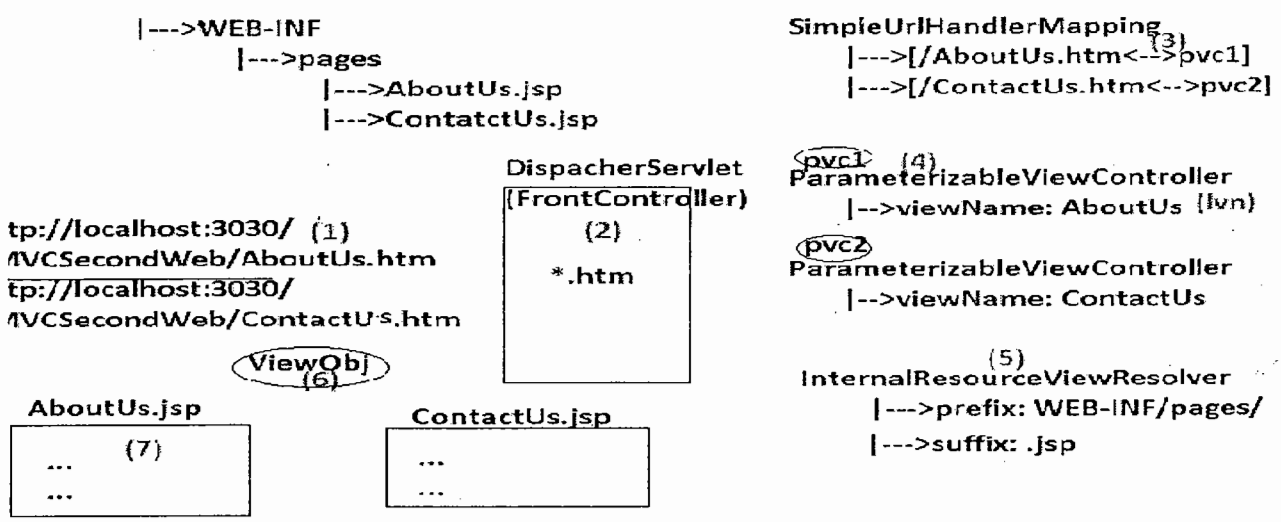


ParameterizableViewController:

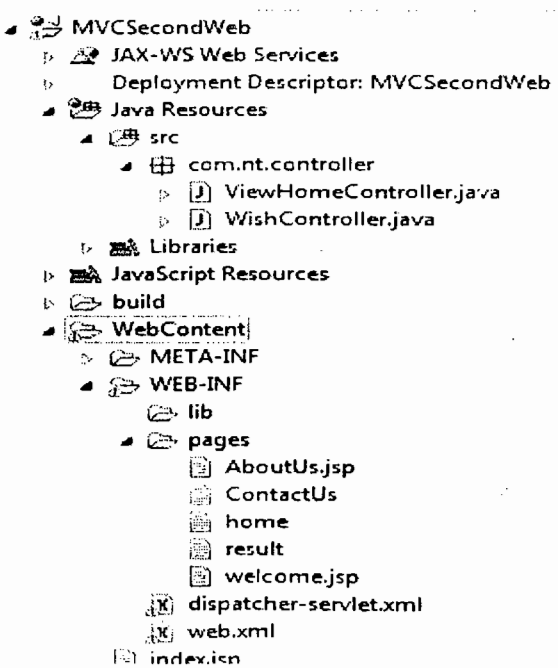
- In order to display webpage without processing request, then instead of taking separate user defined controller class we can configure this class by specifying the destination view name using "viewName" property value. This is useful to display webpage for hyperlink generated request without having any request processing logics

⇨ eg: clicking "AboutUs", "ContactUs" hyperlinks. For multiple request of this kind we need to configure this class for multiple times.

Diagram:




```
SimpleUrlHandlerMapping
        |--->[/Aboutus.htm<->pvc1]
        |--->[/contact.htm<->pvc2]
PVC1
ParameterizableViewController
 |------->ViewName: AboutUs  (logical viewname)
 PVC2
ParameterizableViewController
                |----->ViewName:ContactUs
InternalResourceViewResolver
        |------>Prefix:/WEB-INF/Pages/
        |------->Softfix:.jsp
```



```
HttpServletResponse res) throws Exception {
            // get System date and time
            Calendar cl=Calendar.getInstance();
            //get current hour of the day
            int hour=cl.get(Calendar.HOUR_OF_DAY);
            String msg=null;
            if(hour<12)
                    msg="Good morning";
            else if(hour<16)
                    msg="Good afternoon";
            else if(hour<20)
                    msg="Good evening";
            else
                    msg="Good Night";
            return new ModelAndView("result","wmsg",msg);
    }
}
return new ModelAndView("home");
}
}
```

ViewHomeController.java

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class ViewHomeController extends AbstractController {
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
```

WishController.java

```
package com.nt.controller;
import java.util.Calendar;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class WishController extends AbstractController {
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
```

index.jsp

```
<jsp:forward page="home.htm"/>
```

web.xml

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean id="surl"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="home.htm">vhc</prop>
<prop key="wish.htm">wc</prop>
<prop key="AboutUs.htm">ufv</prop>
```

```
<prop key="ContactUs.htm">ufv</prop>
</props>
</property>
</bean>
<bean id="vhc"  class="com.nt.controller.ViewHomeController"/>
<bean id="wc"  class="com.nt.controller.WishController"/>
<bean id="ufv" class="org.springframework.web.servlet.mvc.UrlFilenameViewController"/>
<!--   <bean id="pvc1"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="AboutUs"/>
</bean>
<bean id="pvc2"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="ContactUs"/>
</bean> -->
<bean id="vr"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

AboutUs.jsp

```
NIT, the poineer in IT Training, got started 13 years back in the year
2003.
```

ContectUs.jsp

```
NIT,
opp Satyam threater
Ammerpet
```

home.jsp

```
<b><center>  Home page</center></b>     <a
href="AboutUs.htm">AboutUs</a>     <a
href="ContactUs.htm">ContactUs</a>
<br>
<a href="wish.htm">Wish Message</a>
```

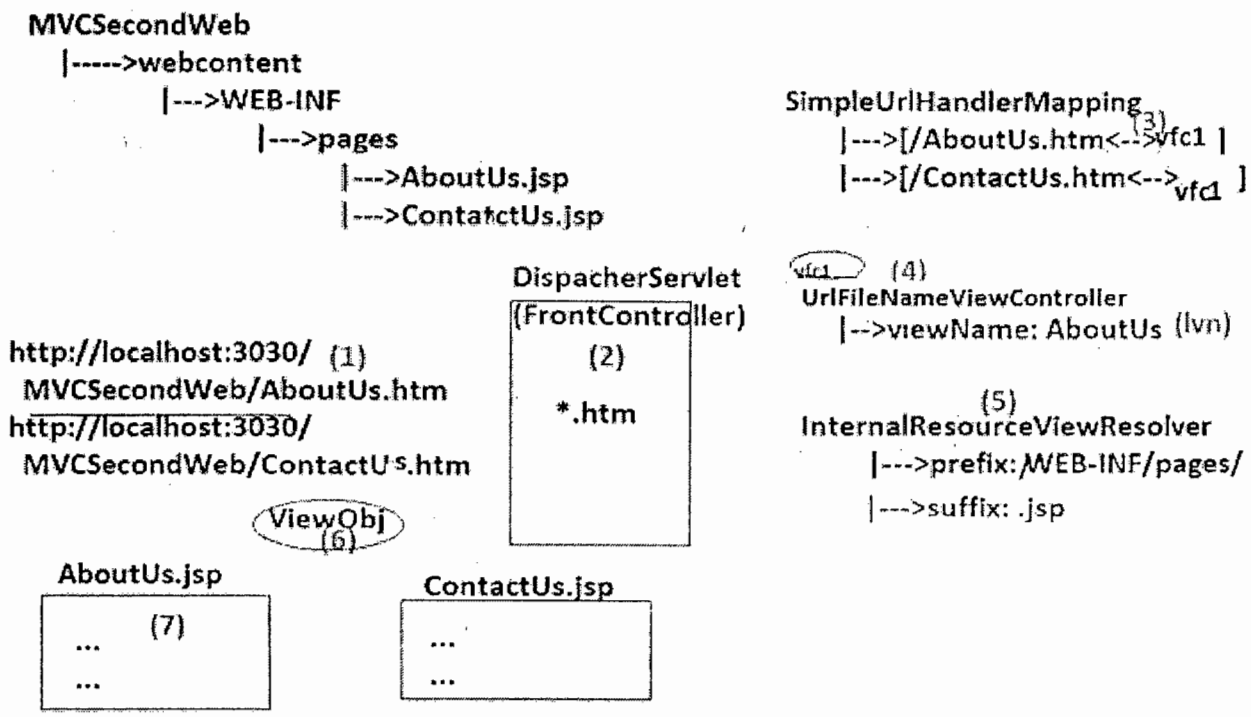
result.jsp

```
<b><center> Result Page</center></b><br>
Wish Message : ${wmsg}.
<br>
<a href="home.htm">home</a>
```

UrlFileNameViewController:

- ⇨ If you want to return the virtual path like (AboutUs) of incoming request url as logical view name to render view/webpages without having any request processing logic then we can use this controller.
- ⇨ This is alternate to using multiple ParameterizableViewController classes configuration to render multiple webpages/views without having any rquest processing for multiple hyperlink generated request like aboutus, contactus and etc....
- ⇨ While configuring this class no need of specifying any property because it takes the virtual path of request url (AboutUS.htm) and return it as logical viewname(AboutUs) automatically.

```
SimpleUrlHandlerMapping
        |------>[/AboutUs.htm<--> UFNVC]
UFNVC
UrlFileNameViewController.
```



Note: If request is there without form data (from hyperlinks) and no request processing is required but displaying views is required, then go for ParameterizableViewController (or) UrlFileNameViewController (like working with AboutUs, ContactUs hyperlink).

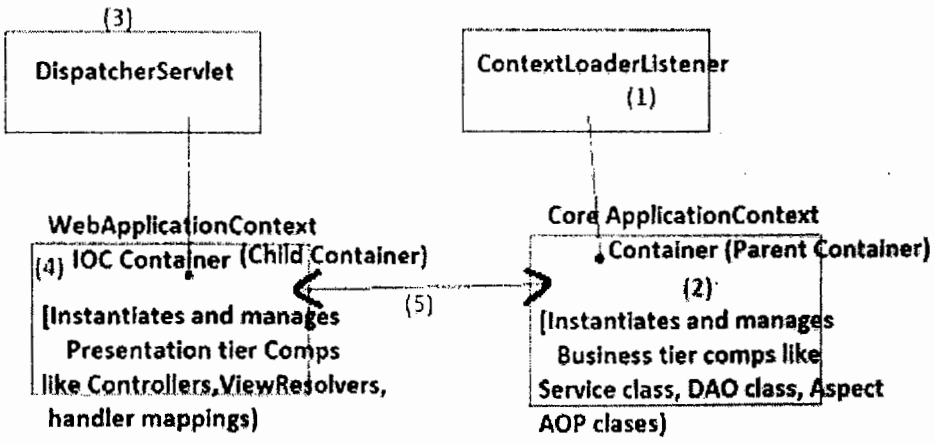
- ⇨ In real-time every spring application will be designed to have 2 IOC containers as parent, child containers because it is recommended to maintain presentation tier components like HandlerMapping. ViewResolver, Controller class in one container (WebApplicationContextContainer) given by DispatcherServlet and business tier component like (service class, DAO class. AOP, Aspect class) in another container (ApplicationContext) IOC container given by **ContextLoaderListner**.

- ⇨ ContexLoaderListner is a spring supplied predefined ServletContextListner which activates ApplicationContext container when servletContext object is created (either during server startup or during the deployment of web application) by taking **contextConfigLocation** contextParamValue as the springbean configuration file.

Code in web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet </servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<context-param>
<param-name>contextConfigLocation</param-name >
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listner>
<listener-class>org.springframework.web.context.ContextLoaderListner</listener-class>
</listner>
</web-app>
```

- If namespaceInitParameter is not configure for DispatcheServlet then the activated webapp context container takes <servlet-name>-servlet.xml as spring bean configure file. (dispatcher-servlet.xml).
- Generally ServletListner object will be created first compare to the servlet object on which load-on-startup is enabled, but some servers creates serlvet object first before Listner object if load-on-startup priority value is 1(or)0.
- We want to see Dispatcher servlets webApplicationContext container as child container to contextLoaderListner's ApplicationContext container in all situations and servers for this it is recommended to enable load-on-startup on DispatcherServlet with priority value other than 1(or)0.
- Since controller classes are managed as beans in child container so they can be injected with service class object/beans managed by parent container...



Since Controller classes are managed as beans in child container so they can be injected with Service class objs/beans managed by Parent Container.

AbstractController:

- ⇨ Take this class to develop controller class when there is no form submission in request but request processing required to display dynamic webpage as output.
- ⇨ Generally for hyperlink generated request to get dynamic content having request processing we should go for AbstractController class.

Usecases:

Getting current user profile in facebook when "viewprofile" link is clicked.
Getting all the jobs that are posted when "current" trends link is clicked.
Getting all employee details when "List of Employees"hyperlink is clicked.

Here no form submission, so there will not be any request wrapping form handling.
Since there is no form submission the handleRequestInternal(-,-) does not take any command object as the argument.

handleRequestInternal(-,-) returns ModelAndView object having logical view name and model Data that has to be displayed as dynamic content.

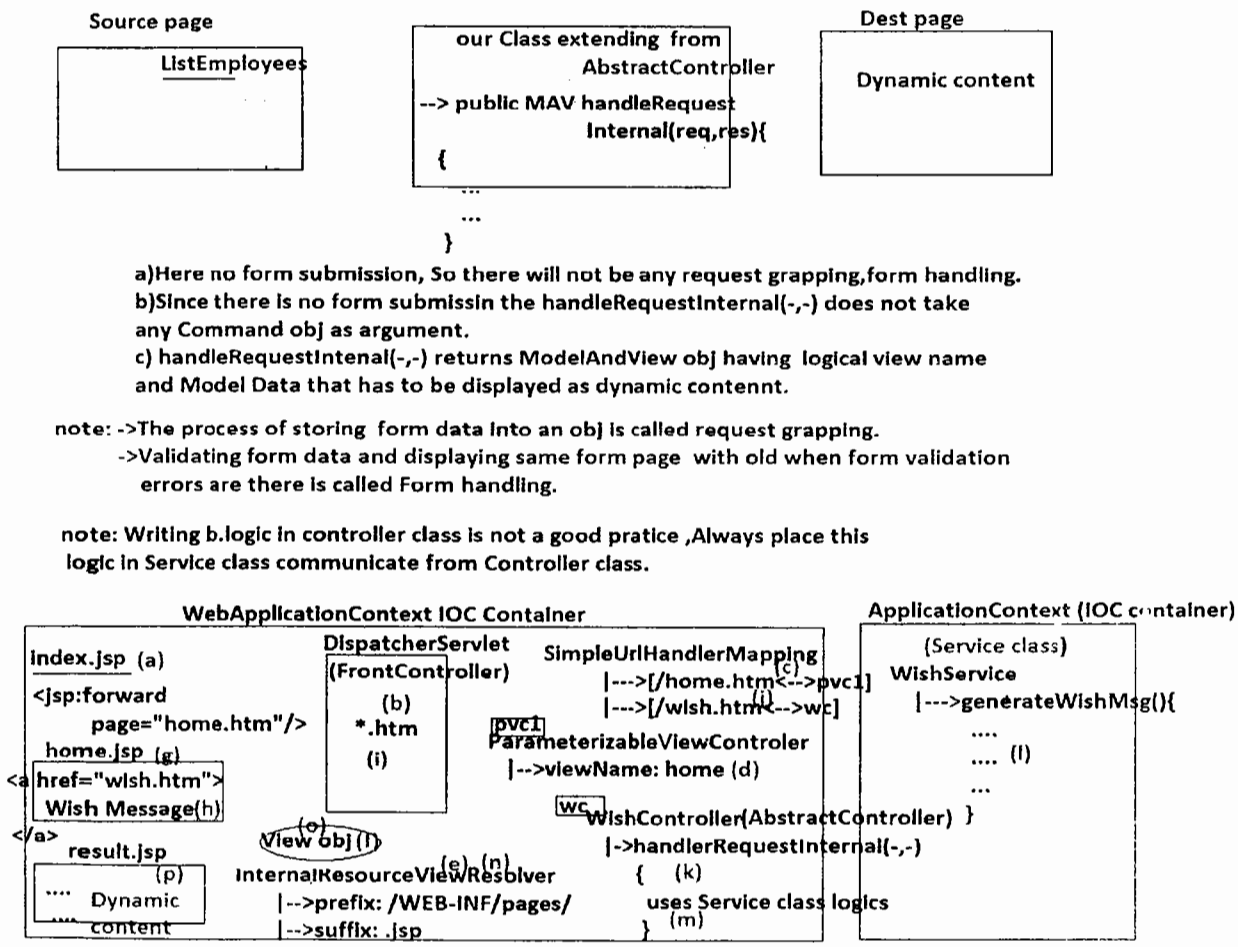
Note:

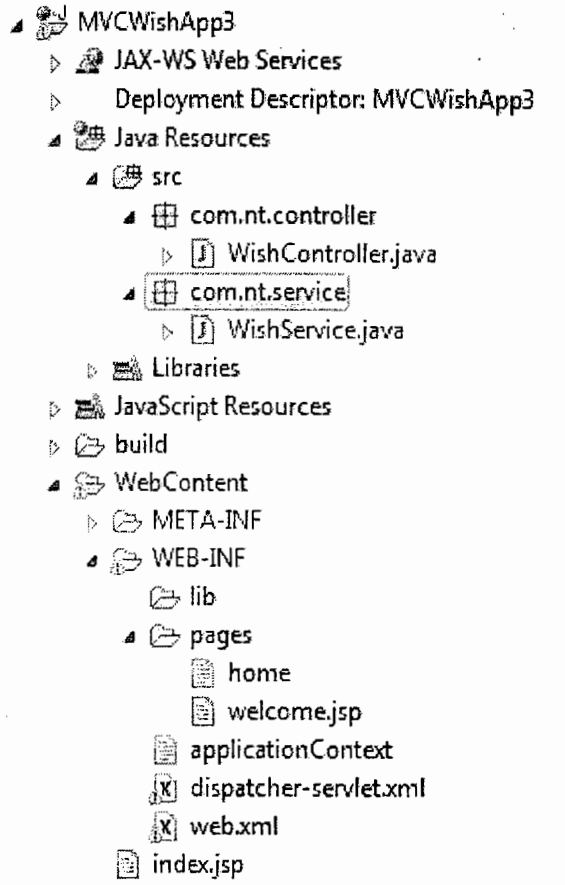
The process of storing form data into an object is called request wrapping.
Validating form data and displaying same form page with old values when form validation errors are there is called Form handling.

Note:

Writing business logic in controller class is not a good practice, always place this logic in service class and communicate from controller class.

Diagram:





WishController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import com.nt.service.WishService;
public class WishController extends AbstractController {
    private WishService service;
    public void setService(WishService service) {
            this.service = service;
    }
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
            //Use SErvice class business logic
            String wmsg=service.generateWishMsg();
            //create and return ModelAndView object
            return new ModelAndView("result","wresult",wmsg);
    }
}
```

WishService.java:

```
package com.nt.service;
import java.util.Calendar;
public class WishService  {
    public String generateWishMsg() {
            //get System date and time
            Calendar cl=Calendar.getInstance();
            //get current hour of the day
            int h=cl.get(Calendar.HOUR_OF_DAY);
            String msg=null;
            if(h<=12)
                    msg="Good morning";
            else if(h<=16)
                    msg="GoodAfter noon";
            else if(h<=20)
                    msg="Good Evening";
            else
                    msg="Good Night";
            return msg;
                    }
}
```

index.jsp:

```
<jsp:forward page="home.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean id="surl"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property          name="mappings">
<props>
<prop key="/home.htm">pvc1</prop>
<prop key="/wish.htm">wc</prop>
</props>
</property>
</bean>
<bean id="wc"  class="com.nt.controller.WishController">
<property name="service" ><ref parent="wService"/></property>
</bean>
<bean id="pvc1"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="home"/>
</bean>
<bean id="vr"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="wService"  class="com.nt.service.WishService"/>
</beans>
```

result.jsp:

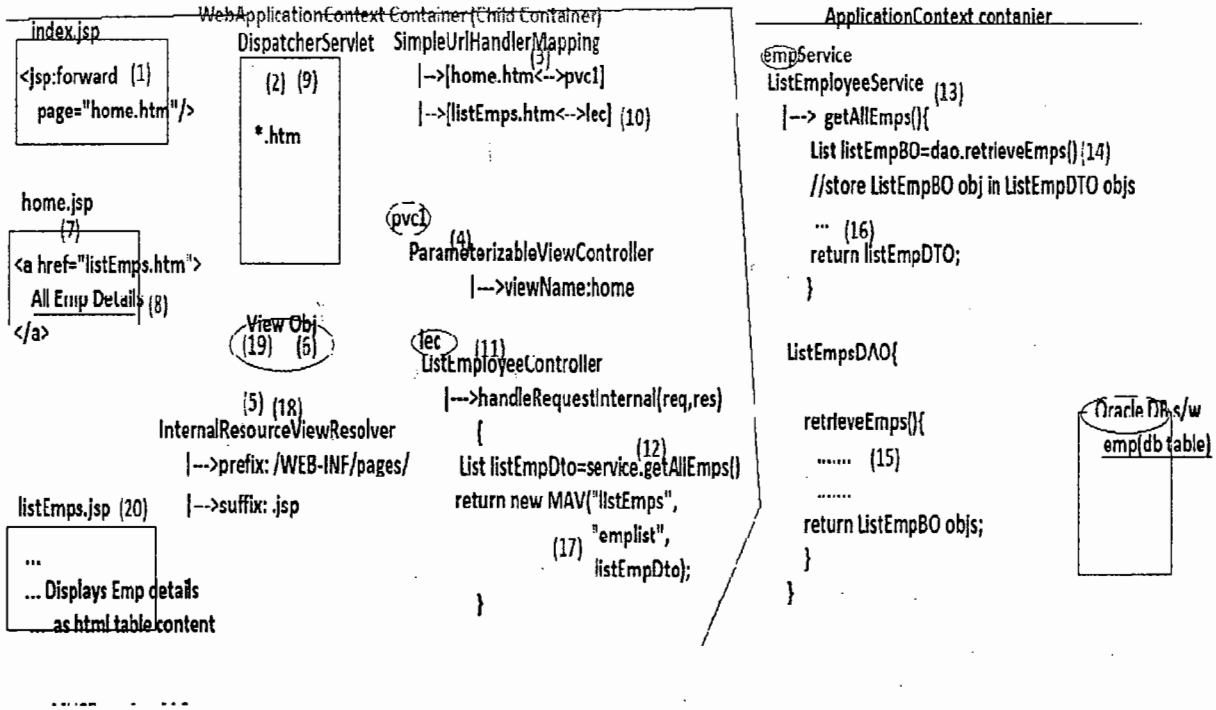
```
<h1><center>Result Page</center></h1>
<br><br>
<font color="red"> ${wresult}</font>
<br>
<a href="home.htm">home</a>
```

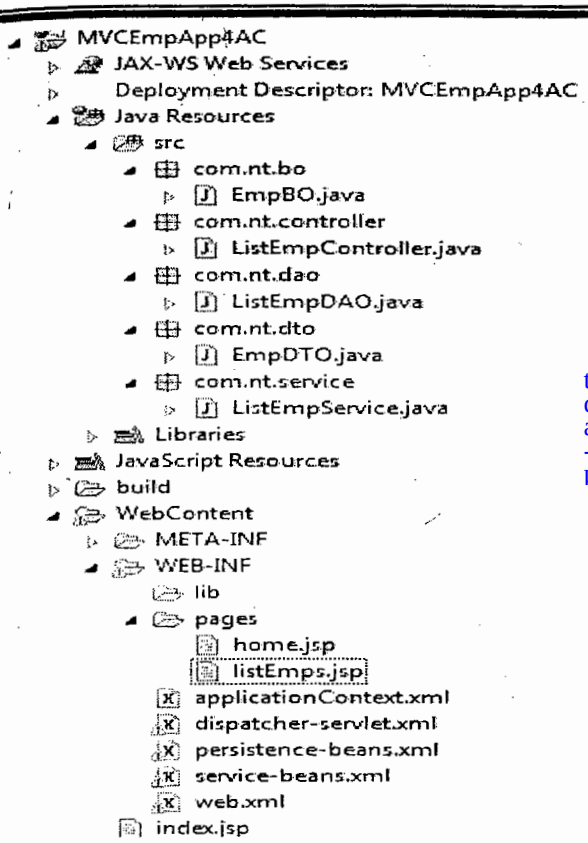
home.jsp:

```
<H1><center> HOME PAGE</center></H1>
<br>
<br>
<a href="wish.htm">Wish Message</a>
```

⇨ In real time beans of service layer will be configureed with separate xml(service-beans.xml) file, similarly the beans of persistence layer will be configured in separate xml file and all this xml will be imported in main applicationContext.xml.

Diagram:





the jars add through pom.xml will be moved automatically to web-inf/lib folder on deployment where as jars add to build path will not be moved to web-inf/lib folder automatically for this we need to use deployment assembly option
--right click on project----arrow properties----deployment assembaly --add java build path entries----select jar----finish----apply----ok

Jars:- MVC library+DAO library+JSTL.jars+standard.jar

EmpBO.java:

```
package com.nt.bo;
public class EmpBO {
    private int no;
    private String name,job;
    private int salary;
            public EmpBO() {
            System.out.println("EmpBO:0-param constructor");
    }
    public EmpBO(int no, String name, String job, int salary) {
            System.out.println("EmpBO:4-param constructor");
            this.no = no;
            this.name = name;
            this.job = job;
            this.salary = salary;
    }
    public int getNo() {
            return no;
    }
    public void setNo(int no) {
            this.no = no;
    }
    public String getName() {
            return name;
    }
    public void setName(String name) {
            this.name = name;
```

```
        }
        public String getJob() {
                return job;
        }
        public void setJob(String job) {
                this.job = job;
        }
        public int getSalary() {
                return salary;
        }
        public void setSalary(int salary) {
                this.salary = salary;
        }
    }
```

ListEmpDAO.java:

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import com.nt.bo.EmpBO;
public class ListEmpDAO {
    private static final String  GET_ALL_EMP_DETAILS="SELECT EMPNO,ENAME,JOB,SAL
FROM  EMP";
    private JdbcTemplate jt;
    public void setJt(JdbcTemplate jt) {
            this.jt = jt;
    }
    public List<EmpBO> retrieveEmps(){
            List<EmpBO> listEmpBO=(List<EmpBO>)jt.query(GET_ALL_EMP_DETAILS,
                                                                    new EmpRowMapper());
            return listEmpBO;
    }
    private class EmpRowMapper implements RowMapper<EmpBO>{

            @Override
            public EmpBO mapRow(ResultSet rs, int pos) throws SQLException {
          //Place ResultSet object record into BO class object
                    EmpBO empbo=new EmpBO(rs.getInt(1),
                                                                    rs.getString(2),
                                                                    rs.getString(3),
                                                                    rs.getInt(4));
                    return empbo;
            }//mapRow
    }//inner class}//clas
```

EmpDTO.java:

```
package com.nt.dto;
import java.io.Serializable;
public class EmpDTO implements Serializable  {
    private int no;
    private String name,job;
    private int salary;

    public EmpDTO() {
            System.out.println("EmpDTO:0-param constructor");
    }
    public EmpDTO(int no, String name, String job, int salary) {
            System.out.println("EmpDTO:4-param constructor");
            this.no = no;
            this.name = name;
            this.job = job;
            this.salary = salary;
    }

    public int getNo() {
            return no;
    }
    public void setNo(int no) {
            this.no = no;
    }
    public String getName() {
            return name;
    }
    public void setName(String name) {
            this.name = name;
    }
    public String getJob() {
            return job;
    }
    public void setJob(String job) {
            this.job = job;
    }
    public int getSalary() {
            return salary;
    }
public void setSalary(int salary) {
            this.salary = salary;
    }
}
```

ListEmpService.java:

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.EmpBO;
import com.nt.dao.ListEmpDAO;
import com.nt.dto.EmpDTO;
public class ListEmpService {
    private ListEmpDAO dao;
    public void setDao(ListEmpDAO dao) {
            this.dao = dao;
    }
                                        M
    public List<EmpDTO>  getAllEmps(){
            //use DAO class
                                        N
        Q   List <EmpBO> listEmpBO=dao.retrieveEmps();
            // Copy List of BO objects to List of DTO objects
            List <EmpDTO> listEmpDTO=new  ArrayList<EmpDTO>();
            for(EmpBO bo:listEmpBO){
                    EmpDTO empdto=new EmpDTO(bo.getNo(),
    bo.getName(),bo.getJob(),bo.getSalary());
            listEmpDTO.add(empdto);
            }//for
            return listEmpDTO;   R
}//method
}//class
```

ListEmpController.java:

```
package com.nt.controller;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import com.nt.dto.EmpDTO;
import com.nt.service.ListEmpService;
public class ListEmpController extends AbstractController {
     private  ListEmpService service;

    public void setService(ListEmpService service) {
            this.service = service;
    }

    @Override
                                        K
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
            //use service class logics
```

```
        List <EmpDTO> listEmpDTO=service.getAllEmps();
    return new ModelAndView("listEmps","empList",listEmpDTO);
    }
}
```

index.jsp:

```
<jsp:forward page="home.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

service-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="empService"  class="com.nt.service.ListEmpService">
<property name="dao"  ref="empDao"/>
</bean>
</beans>
```

persistence-beans.xm:l

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="empDao"  class="com.nt.dao.ListEmpDAO">
<property name="jt" ref="template"/>
</bean>
</beans>
```

dispatcher-servlet-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-3.0.xsd">
<bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/home.htm">pvc1</prop>  C
<prop key="/listEmps.htm">lec</prop> I
</props>
</property>
</bean>
<bean id="lec" J class="com.nt.controller.ListEmpController">
<property name="service"  ref="empService"/>
</bean>
<bean id="pvc1"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="home"/>      d
</bean>
                                    U--listEmps.jsp
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
                                    e
</bean>
</beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<import resource="persistence-beans.xml"/>
<import resource="service-beans.xml"/>
</beans>
```

home.jsp: F

```
<h1><Center>HOME  PAGE</Center></h1>
<br>
<a href="listEmps.htm"> All Employees</a>
```

G

listEmps.jsp: V

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<table border="1" bgcolor='red'>
<tr bgcolor="blue"><th>eno</th><th>ename</th><th>JOB</th><th>Salary</th></tr>
<c:forEach var="emp" items="${empList}">
<tr bgcolor="cyan">
<td><c:out value="${emp.no}" /></td>
<td><c:out value="${emp.name}" /></td>
<td><c:out value="${emp.job}" /></td>
<td><c:out value="${emp.salary}" /></td>
</tr>
</c:forEach>
</table>
```

==000==

Command Class:

- The java bean class that would be instiantate by spring mvc to store form data or form page data is called command class. The command class properties names and the formpage component name must match. This avoids programmers burden of calling request.getParameter(-) and writing form data to java bean object.

A command class properties can be of any type hold different types of values given by form page.

AbstractCommandController:

- This controller is capable of reading form data of formpage and writes that to command class, i.e it takes care of request graphing/wrapping.
- Use this controller when source page having form submission with no serverside form validation, client side form validation optional and no form handling (no need of displaying same form page with old value if error occurs) and request wrapping is required, destination page should be static or dynamic page that should come after request processing. (Request processing is there)When we develop controller class extending from AbstractCommandController we must implement handle(-,-,-) method

public ModelAndView handle(HttpServletRequest req, HttpServletResponse res, Object cmd, BindException be)

Form submission
No serverside form validation, optional client side form validations.
No form handling.

Controller:

- Recieves form data and stores in command class object (request graphing/wrapping)

Process the request by taking support of service, DAO classes.
Returns ModelAndView object logical view name and model data (result)

Dest Page:
Static or dynamic content that comes after request processing.

Usecase:
Searching employee details based on given no details or 1/2/3.....details in a form.
While configuring controller class whose type is AbstractCommandController we must specify property "commandClass" having fully qualified command class name as the value.

Diagram:

Source page

form submission

-> No Server side form validations, optional client Side form validations

->No Form handling

Controller

(AbstractCommandController)

-> handle(-,-,-,-)

(request wrapping to Command class)

Dest page

Static/Dyanamic content (throuhg request Procesing)

Source page:

-> Form with submit button or java script based form submission having optional client side form validations

->Since form is there request wrapping/graphing is required

->No Form handling

Controller

=> Recives form data and strores in command class obj (request wrapping/graphing)

=> Process the request by taking support of Service,DAO classes

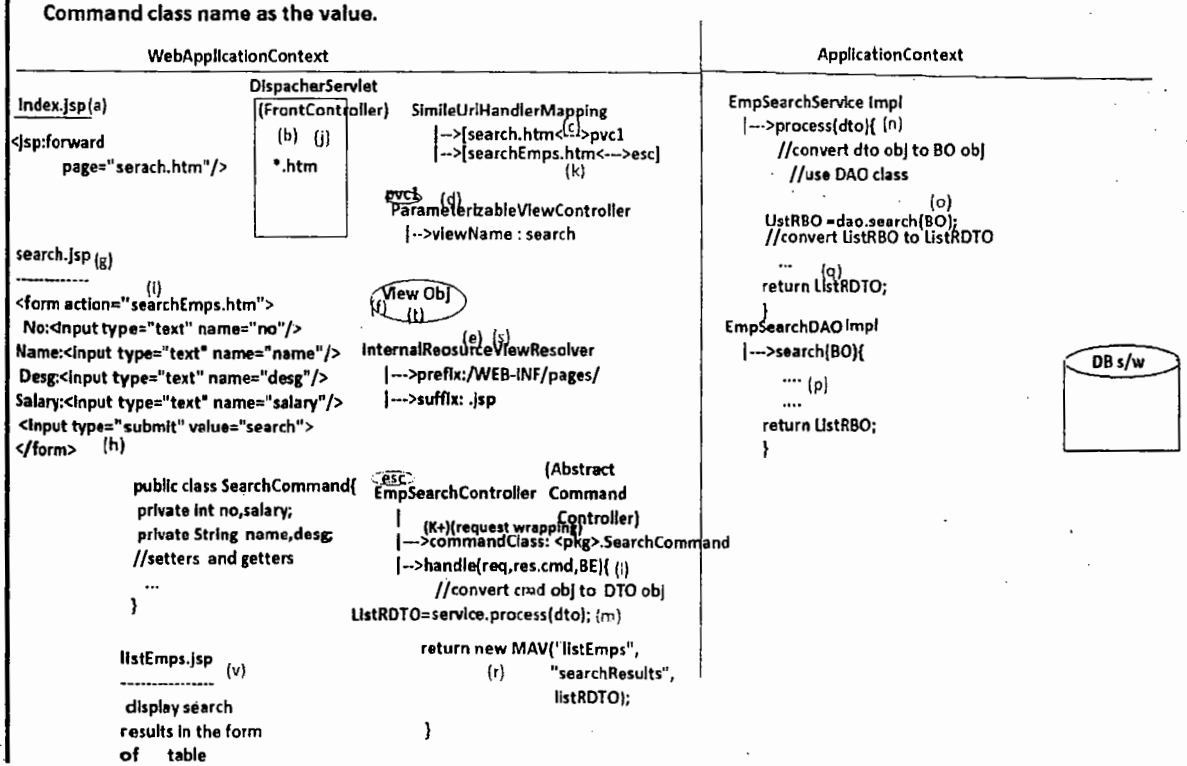
=> Returns ModelAndView obj having logical view name and Model data (results)

Dest page

=> Static /dyanamic content that comes after request processing.

Usecase: Searching employee details based given no details or 1/2/3.. details in a form .

=>While cfg Controller class whose type is AbstractCommandController we must specify property "commandClass" having fully qualified Command class name as the value.



- MVCEmpSearchApp5ACC
  - JAX-WS Web Services
  - Deployment Descriptor: MVCEmpSearchApp
  - Java Resources
    - src
      - com.nt.bo
        - SearchBO.java
        - SearchResultBO.java
      - com.nt.command
        - SearchCommand.java
      - com.nt.controller
        - EmpSearchController.java
      - com.nt.dao
        - EmpSearchDAO.java
        - EmpSearchDAOImpl.java
      - com.nt.dto
        - SearchDTO.java
        - SearchResultDTO.java
      - com.nt.service
        - EmpSearchService.java
        - EmpSearchServiceImpl.java
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
      - pages
      - aop-beans.xml

Note: Since AbstractCommondController,SimpleFormController class are removed from spring 4.x in support to annotation. so we use spring 3.2.6 + servlet 3.0.1 in the development of application.

SearchBO.java:

```
package com.nt.bo;
public class SearchBO {
    private int no;
    private String name,desg;
    private int salary;
    public SearchBO() {
    }
    public SearchBO(int no, String name, String desg, int salary) {
        this.no = no;
        this.name = name;
        this.desg = desg;
        this.salary = salary;
    }
    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
```

```
    public String getName() {
            return name;
    }
    public void setName(String name) {
            this.name = name;
    }
    public String getDesg() {
            return desg;
    }
    public void setDesg(String desg) {
            this.desg = desg;
    }
    public int getSalary() {
            return salary;
    }
    public void setSalary(int salary) {
            this.salary = salary;
    }
}
```

SearchResultBO.java:

```
package com.nt.bo;
public class SearchResultBO {
    private int no;
    private String name,desg;
    private int salary,deptno,mgrno;
    public SearchResultBO() {
            }
    public SearchResultBO(int no, String name, String desg, int salary,
                    int deptno, int mgrno) {
            this.no = no;
            this.name = name;
            this.desg = desg;
            this.salary = salary;
            this.deptno = deptno;
            this.mgrno = mgrno;
    }
    public int getNo() {
            return no;
    }
    public void setNo(int no) {
            this.no = no;
    }
    public String getName() {
            return name;
    }
```

```
    public void setName(String name) {
            this.name = name;
    }
    public String getDesg() {
            return desg;
    }
    public void setDesg(String desg) {
            this.desg = desg;
    }
    public int getSalary() {
            return salary;
    }
    public void setSalary(int salary) {
            this.salary = salary;
    }
    public int getDeptno() {
            return deptno;
    }
    public void setDeptno(int deptno) {
            this.deptno = deptno;
    }

    public int getMgrno() {
            return mgrno;
    }
    public void setMgrno(int mgrno) {
            this.mgrno = mgrno;
    }
}
```

EmpSearchDAO.java:

```
package com.nt.dao;
import java.util.List;
import com.nt.bo.SearchBO;
import com.nt.bo.SearchResultBO;
public interface EmpSearchDAO {
    public List<SearchResultBO> search(SearchBO  sbo);
}
```

EmpSearchDAOImpl.java:

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import com.nt.bo.SearchBO;
import com.nt.bo.SearchResultBO;
public class EmpSearchDAOImpl implements EmpSearchDAO {
```

```
    private static final String EMP_SEARCH_QUERY="select
empno,ename,job,sal,mgr,deptno  from emp "+ " where (empno is not null and empno=? )"
  + "or (ename is  not null and ename like ?)"
  + "or (job is not null and job like ?)"
  + "or (sal is not null  and sal=?)";

  private JdbcTemplate jt;
    public void setJt(JdbcTemplate jt) {
        this.jt = jt;
    }

  @Override
    public List<SearchResultBO> search(SearchBO sbo) {
  List <SearchResultBO> listRBO=jt.query(EMP_SEARCH_QUERY, newEmpRowMapper(),
  sbo.getNo(),
  sbo.getName(),
  sbo.getDesg(),
  sbo.getSalary());
        return listRBO;
    }

    private static final class EmpRowMapper implements RowMapper<SearchResultBO>{
        @Override
        public SearchResultBO mapRow(ResultSet rs, int pos) throws SQLException {
            //copy ResultSet record into SearchResultBO object
            SearchResultBO  rbo=new SearchResultBO();
            rbo.setNo(rs.getInt(1));
            rbo.setName(rs.getString(2));
            rbo.setDesg(rs.getString(3));
            rbo.setSalary(rs.getInt(4));
            rbo.setMgrno(rs.getInt(5));
            rbo.setDeptno(rs.getInt(6));
            return rbo;
        }
    }
  }
```

SearchDTO.java:

```
  package com.nt.dto;
  import java.io.Serializable;
  public class SearchDTO implements Serializable {
    private int no;
    private String name,desg;
    private int salary;
  public SearchDTO() {
        }
  public SearchDTO(int no, String name, String desg, int salary) {
            this.no = no;
```

```
        this.name = name;
        this.desg = desg;
        this.salary = salary;
    }
    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDesg() {
        return desg;
    }
    public void setDesg(String desg) {
        this.desg = desg;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }}
```

SearchResultDTO.java:

```
package com.nt.dto;
import java.io.Serializable;
public class SearchResultDTO implements Serializable {
    private int no;
    private String name,desg;
    private int salary,mgrno,deptno;
    public SearchResultDTO() {
    }
    public SearchResultDTO(int no, String name, String desg, int salary,
                int mgrno, int deptno) {
                this.no = no;
        this.name = name;
        this.desg = desg;
        this.salary = salary;
        this.mgrno = mgrno;
        this.deptno = deptno;
    }
```

```
    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDesg() {
        return desg;
    }
    public void setDesg(String desg) {
        this.desg = desg;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
    public int getMgrno() {
        return mgrno;
    }
    public void setMgrno(int mgrno) {
        this.mgrno = mgrno;
    }
    public int getDeptno() {
        return deptno;
    }
    public void setDeptno(int deptno) {
        this.deptno = deptno;
    }
}
```

EmpSearchService.java:

```
package com.nt.service;
import java.util.List;
import com.nt.dto.SearchDTO;
import com.nt.dto.SearchResultDTO;
public interface EmpSearchService {
    public List<SearchResultDTO> process(SearchDTO sdto);
}
```

EmpSearchServiceImpl.java:

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.SearchBO;
import com.nt.bo.SearchResultBO;
import com.nt.dao.EmpSearchDAO;
import com.nt.dto.SearchDTO;
import com.nt.dto.SearchResultDTO;
public class EmpSearchServiceImpl implements EmpSearchService {
    private EmpSearchDAO dao;
    public void setDao(EmpSearchDAO dao) {
            this.dao = dao;
    }
    @Override
    public List<SearchResultDTO> process(SearchDTO sdto) {
            //Convert SearchDTO object to SearchBO object
            SearchBO  sbo=new SearchBO();
            sbo.setNo(sdto.getNo());
            sbo.setName(sdto.getName());
            sbo.setDesg(sdto.getDesg());
            sbo.setSalary(sdto.getSalary());
            //Use DAO
            List<SearchResultBO> listRBO=dao.search(sbo);
            //Convert ListRBO to ListRDTO  object
            List<SearchResultDTO> listRDTO=new ArrayList<SearchResultDTO>();
            for(SearchResultBO rbo:listRBO){
                    SearchResultDTO rdto=new SearchResultDTO();
                    rdto.setNo(rbo.getNo());
                    rdto.setName(rbo.getName());
                    rdto.setDesg(rbo.getDesg());
                    rdto.setSalary(rbo.getSalary());
                    rdto.setMgrno(rbo.getMgrno());
                    rdto.setDeptno(rbo.getDeptno());
                    listRDTO.add(rdto);
            }//for
            return listRDTO;
    }//process
}//class
```

SearchCommand.java:

```
package com.nt.command;
public class SearchCommand {
    private int no;
    private String name,desg;
    private int salary;
    public SearchCommand() {
      System.out.println("SearchCommand:0-param constructor");
    }
```

```
    public int getNo() {
            return no;
    }
    public void setNo(int no) {
            this.no = no;
    }
    public String getName() {
            return name;
    }
    public void setName(String name) {
            this.name = name;
    }
    public String getDesg() {
            return desg;
    }
    public void setDesg(String desg) {
            this.desg = desg;
    }
    public int getSalary() {
            return salary;
    }
    public void setSalary(int salary) {
            this.salary = salary;
    }
}
```

EmpSearchController.java:

```
package com.nt.controller;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractCommandController;
import com.nt.command.SearchCommand;
import com.nt.dto.SearchDTO;
import com.nt.dto.SearchResultDTO;
import com.nt.service.EmpSearchService;
public class EmpSearchController extends AbstractCommandController {
    private EmpSearchService service;
    public void setService(EmpSearchService service) {
            this.service = service;
    }
    @Override
    protected ModelAndView handle(HttpServletRequest req,
                    HttpServletResponse res, Object cmd, BindException be)
                    throws Exception {
            //get Command object
```

```
        SearchCommand command=(SearchCommand)cmd;
        //convert Command object into DTO object
        SearchDTO  dto=new SearchDTO();
        dto.setNo(command.getNo());
        dto.setName(command.getName());
        dto.setDesg(command.getDesg());
        dto.setSalary(command.getSalary());
        //Use Service class
        List <SearchResultDTO> listRDTO=service.process(dto);
        return new ModelAndView("listEmps","searchResults",listRDTO);
    }
}
```

index.jsp:

```
<jsp:forward page="search.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
```

```
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

service-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="esService"  class="com.nt.service.EmpSearchServiceImpl">
    <property name="dao"  ref="esDao"/>
    </bean></beans>
```

persistence-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">
   <bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
   <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
   <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
   <property name="username" value="scott"/>
   <property name="password" value="tiger"/>
   </bean>
   <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
   <property name="dataSource" ref="dataSource"/>
   </bean>
   <bean id="esDao"  class="com.nt.dao.EmpSearchDAOImpl">
   <property name="jt" ref="template"/>
   </bean>
   </beans>
```

dispatcher-servlet.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-3.0.xsd">
   <bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="mappings">
   <props>
   <prop key="/search.htm">pvc1</prop>
   <prop key="/searchEmps.htm">esc</prop>
   </props>
   </property>
   </bean>
   <bean id="esc"  class="com.nt.controller.EmpSearchController">
   <property name="commandClass" value="com.nt.command.SearchCommand"/>
   <property name="service"  ref="esService"/>
   </bean>
   <bean id="pvc1"  class="
   org.springframework.web.servlet.mvc.ParameterizableViewController">
   <property name="viewName"  value="search"/>
   </bean>
   <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
   <property name="prefix"  value="/WEB-INF/pages/"/>
   <property name="suffix"  value=".jsp"/>
   </bean>
   </beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<import resource="persistence-beans.xml"/>
<import resource="service-beans.xml"/>
<import resource="aop-beans.xml"/>
</beans>
```

aop-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
<beanid="txMgr"class="org.springframework.jdbc.datasource.DataSourceTransactionManag
er">
<property name="dataSource"  ref="dataSource"/>
</bean>
<tx:advice id="txAdvice" transaction-manager="txMgr">
<tx:attributes>
<tx:method name="process"  propagation="REQUIRED"
                                                read-only="true"/>
</tx:attributes>
</tx:advice>
<aop:config>
<aop:pointcut id="ptc1"
                        expression="execution(*
com.nt.service.EmpSearchService.process(..))"/>
<aop:advisor advice-ref="txAdvice" pointcut-ref="ptc1"/>
</aop:config>
</beans>
```

search.jsp

```
<center><H1>Search Employees</H1></center>
<form action="searchEmps.htm"  method="POST">
   Emp no:  <input type="text" name="no"><br>
   Emp Name:  <input type="text" name="name"><br>
   Emp Desg:  <input type="text" name="desg"><br>
   Emp Salary:  <input type="text" name="salary"><br>
<input type="submit"  value="Search"></form>
```

listEmps.jsp:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>
<center><h1>Search Results</h1></center>
<c:if test="${not empty searchResults}">
<table>
<tr>
<th>Emp No</th>
<th>Emp Name</th>
<th>Salary</th>
<th>Desg</th>
<th>Dept no</th>
<th>Manger No</th>
</tr>
<c:forEach items="${searchResults}"  var="result">
<tr>
<td><c:out value="${result.no}"/></td>
<td><c:out value="${result.name}"/></td>
<td><c:out value="${result.salary}"/></td>
<td><c:out value="${result.desg}"/></td>
<td><c:out value="${result.deptno}"/></td>
<td><c:out value="${result.mgrno}"/></td>
</tr>
</c:forEach>
</table>
</c:if>
```

Internal flow of abstract command controller:-

Dispatcher Servlet

```
(h)
controller.handle (a)
Request(req,res);
```

```
public class AbstractCommandController implements Controller{
                    (b)
public MAV handlerRequest(req,res){
->create command class obj writes the received from data to the obj. (c)
->creates an Empty BinderException class ob.
->calls handler(-,-,-,-)with req,res,command obj,BindException obj. (d)
-> return MAV obj given by handler(-,-,-)call (g)
  }
->public abstract MAV handle(req,res,Object cmd,BindException)
  }
public class EmpSearchController extends
AbstractCommandController {
 public MAV handler(req,res,Object cmd,BindException be){
.........................
.........................
.........................
return new MAV(-,-,-);(f)
  }
 }
```

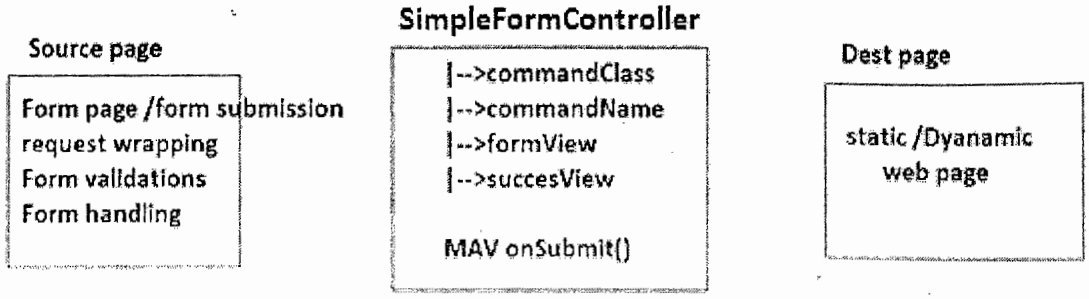
- The moment we enable the Transaction management the underlying database software will be applied with read commited isolation level. i.e means it allows to read only committed data from database.
- For every request given our AbstractCommandController class, one object of command class will be created for request wrapping.
- DispatcherServlet always gets our controller class objects from webApplicationContext container by using ctx .getBean (-) and calls handleRequest(req, res) on that object. If not available then the super class handleRequest(-,-) executes.
- DispatcherServlet doesn't create command class object. The handleRequest(-,-) of superclass to our controller class like AbstractCommandController/SimpleFormController .... Will create this command class object.

<u>SimpleFormController</u>:

- It is a multipurpose controller which can be used for form submission, form validations, request wrapping, solving double posting and etc....

- If our source page having form page with form validation, form handling and looking to get Dest page either as static/dynamic page having request processing then use SimpleFormController.

Diagram:



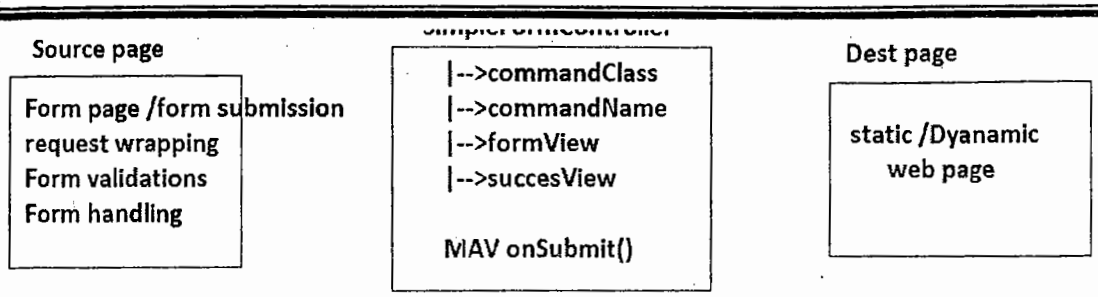
The SimpleFormController displays form page based on "formView" value if the request is initialphase request (request method:GET) i.e there is no need taking seperate Parameterizable ViewController for launching form page.

The same controller becomes ready to process the request if the request is PostBack request (request method: POST)

1) DispacherServlet calls handleRequest(req,res) on our SimpleFormController class obj
2) Since not avaible super class (SimpleFormController) handleRequst(-,-) method executes
3) The handleRequest(-,-) checks weather request is initialphase request or post back request

Initial Phase Request (if req.getMethod.equals("GET")){
a) Creates an empty Command class obj
b) Keeps Command class obj either in Session/request scope
c) gather formView property value and returns it as logical View name to DispatherServlet
d) ViewResolver takes lvn (logical view name) and gives View obj to render form page to browser .This form page can display the initial values by collecting from Command obj
}

Post back Request (if req.getMethod.equals("POST")){
a) creates /Locates Comand class obj and writes form data to it (request wrapping)
b)creates an empty BindException obj
c) calls onSubmit() having req,res,command,BinException objs
d) onSubmit() our Controller class will process the request and returns MAV obj having logical view name
e) ViewResolver takes lvn and gives view obj to render result page.



The SimpleFormController displays form page based on "formView" value if the request is initialphase request (request method:GET) i.e there is no need taking seperate Parameterizable ViewController for launching form page.

The same controller becomes ready to process the request if the request is PostBack request (request method: POST)

1) DispacherServlet calls handleRequest(req,res) on our SimpleFormController class obj
2) Since not avaible super class (SimpleFormController) handleRequst(-,-) method executes
3) The handleRequest(-,-) checks weather request is initialphase request or post back request

Initial Phase Request (if req.getMethod.equals("GET")){
a) Creates an empty Command class obj
b) Keeps Command class obj either in Session/request scope
c) gather formView property value and returns it as logical View name to DispatherServlet
d) ViewResolver takes lvn (logical view name) and gives View obj to render form page to browser .This form page can display the initial values by collecting from Command obj
}

Post back Request (if req.getMethod.equals("POST")){
a) creates /Locates Comand class obj and writes form data to it (request wrapping)
b)creates an empty BindException obj
c) calls onSubmit() having req,res,command,BinException objs
d) onSubmit() our Controller class will process the request and returns MAV obj having logical view name
e) ViewResolver takes lvn and gives view obj to render result page.

SimpleFormController
|-------->CommandClass
|-------->CommandName
|--------->formView:form page name
|--------->successView: Result page name
|--------->sessionForm and etc.....
|-------->true--->keep cmd object in session object.
|--------->false---->keep cmd object in requestScope.

Methods
|------>public MAV onSubmit (Object cmd)
|------>public MAV onSubmit (Object cmd, BindException be)
|------>public MAV onSubmit (req,res, Object cmd, BindException be)

4 param method internally call **2 param method** and **2 param method** internally calls **1 param method**

Note:

- ⇨ working with SimpleformController our formpage request method must be "POST".
- ⇨ In form handling we need to display the same page with old values when form validation error are there. In this process we need to write lots of additional code for this if we work with traditional HTML pages.
- ⇨ If we use spring supplies tag library tags then automatically it take care of this process which is nothing but reading old values from command object and displaying in the form page.

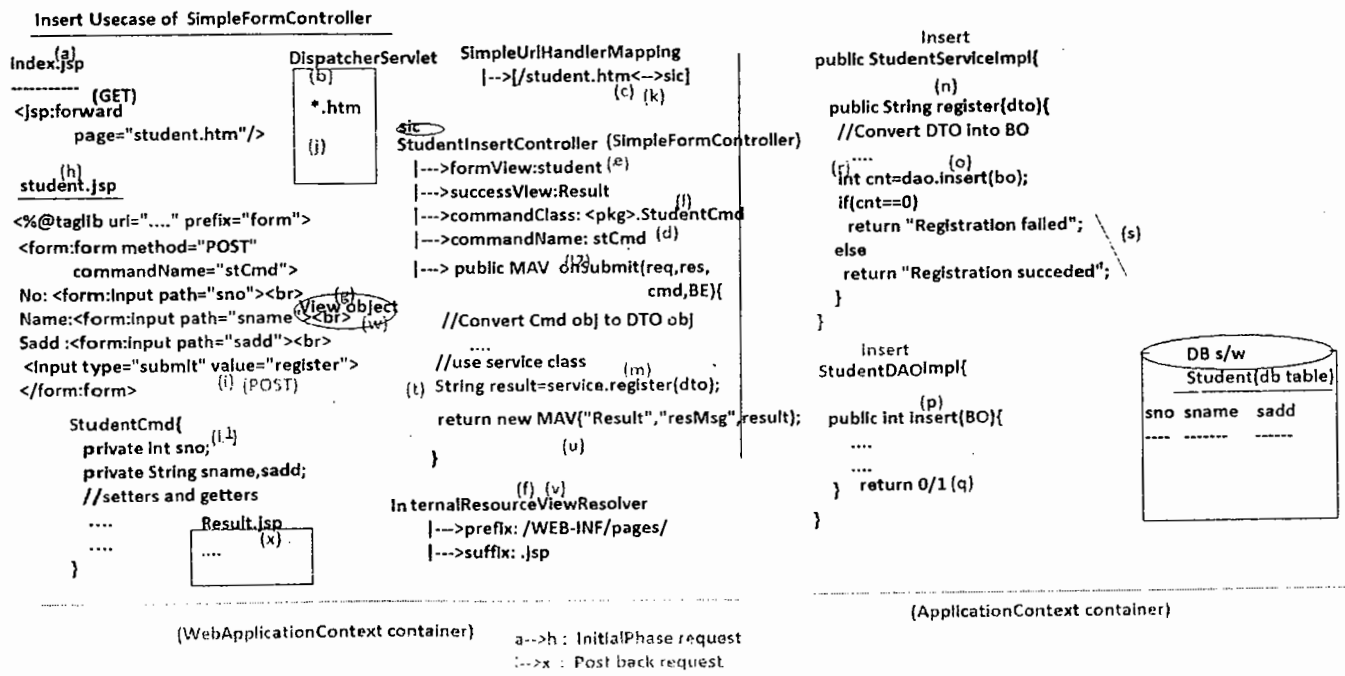
Spring gives 2 jsp tag libraries to develop that jsps of web application:

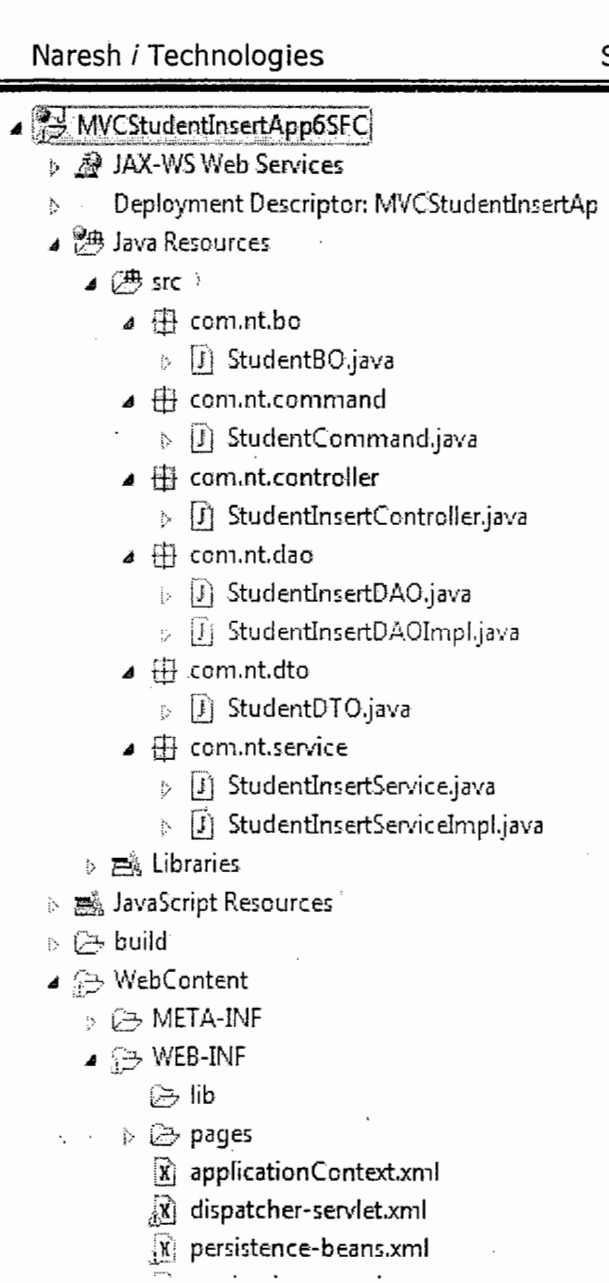
Generic jsp tag library
- |------->general tags
- |------->tld File=spring.tld
- |-------->taglib uri:http//www.springframework.org/tags

Form jsp tag library
- |--------> form tags
- |--------->tld File: Spring_form.tld
- |-------->taglib uri:http//www.springframewrok.org/tags/form

Note:
AbstractCommandController,SimpleFormController classes are deprecatedin spring 3.x and removed from spring 4.x in support to annotation based controllers.

Diagram:





Service-beans.xml

StudentBO.java:

```
package com.nt.bo;
    public class StudentBO {
        private int sno;
        private String sname,sadd;
        public int getSno() {
                return sno;
        }
        public void setSno(int sno) {
                this.sno = sno;
        }
        public String getSname() {
                return sname;
        }
        public void setSname(String sname) {
                this.sname = sname;
        }
        public String getSadd() {
```

```
            return sadd;
        }
        public void setSadd(String sadd) {
            this.sadd = sadd;
        }
    }
```

StudentInsertDAO.java:

```
package com.nt.dao;
import com.nt.bo.StudentBO;
public interface StudentInsertDAO {
    public int insert(StudentBO sbo);
}
```

StudentInsertDAOImpl.java:

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
import com.nt.bo.StudentBO;
public class StudentInsertDAOImpl implements StudentInsertDAO {
    private static final String  INSERT_STUDENT_QRY="INSERT INTO STUDENT
values(?,?,?)";
    private JdbcTemplate jt;
    public void setJt(JdbcTemplate jt) {
        this.jt = jt;
    }
    @Override
    public int insert(StudentBO sbo) {
    int cnt=jt.update(INSERT_STUDENT_QRY,sbo.getSno(),
    sbo.getSname(),
                                                    sbo.getSadd());
        return cnt;
    } }
```

StudentDTO.java:

```
package com.nt.dto;
import java.io.Serializable;
public class StudentDTO implements Serializable {
    private int sno;
    private String sname,sadd;
    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
}
```

```
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
}
```

StudentInsertService.java:

```
package com.nt.service;
import com.nt.dto.StudentDTO;
public interface StudentInsertService {
    public String  register(StudentDTO dto);
}
package com.nt.service;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentInsertDAO;
import com.nt.dto.StudentDTO;
public class StudentInsertServiceImpl implements StudentInsertService{
    private StudentInsertDAO dao;
    public void setDao(StudentInsertDAO dao) {
        this.dao = dao;
    }
    @Override
    public String register(StudentDTO dto) {
        // Convert DTO object BO object
        StudentBO sbo=new StudentBO();
        sbo.setSno(dto.getSno());
        sbo.setSname(dto.getSname());
        sbo.setSadd(dto.getSadd());
        //Use DAO
        int cnt=dao.insert(sbo);
        if(cnt==0)
            return " Student Regisration Failed";
        else
            return " Student Regisration Succeded with no"+sbo.getSno();
    }
}
```

StudentInsertController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.nt.command.StudentCommand;
import com.nt.dto.StudentDTO;
import com.nt.service.StudentInsertService;
public class StudentInsertController extends SimpleFormController {
    private StudentInsertService  service;
    public void setService(StudentInsertService service) {
            this.service = service;
    }
    @Override
    protected ModelAndView onSubmit(HttpServletRequest request,
                    HttpServletResponse response, Object command, BindException errors)
                    throws Exception {
            //type cast cmd object
            StudentCommand cmd=(StudentCommand)command;
            //Convert Command object to DTO object
            StudentDTO  dto=new StudentDTO();
            dto.setSno(cmd.getSno());
            dto.setSname(cmd.getSname());
            dto.setSadd(cmd.getSadd());
            //Use Service class
            String result=service.register(dto);
            //return ModelAndView object
            return new ModelAndView(this.getSuccessView(),"resMsg",result);
    }
}
```

StudentCommand.java:

```
package com.nt.command;
public class StudentCommand {
    private int sno=101;
    private String sname="raja",sadd="hyd";
    public int getSno() {
            return sno;
    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
```

```
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
            this.sadd = sadd;
    }
}
```

index.jsp:

```
<jsp:forward page="student.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/student.htm">sic</prop>
</props>
```

```
</property>
</bean>
<bean id="sic"  class="com.nt.controller.StudentInsertController">
<property name="formView" value="student"/>
<property name="successView" value="Result"/>
```

```
<property name="commandClass" value="com.nt.command.StudentCommand"/>
<property name="commandName" value="stCmd"/>
<property name="sessionForm" value="true"/>
<property name="service"  ref="stService"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<import resource="persistence-beans.xml"/>
<import resource="service-beans.xml"/>
</beans>
```

service-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="stService"  class="com.nt.service.StudentInsertServiceImpl">
<property name="dao"  ref="stDao"/>
</bean>
</beans>
```

persistence-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
```

```
<bean id="stDao"  class="com.nt.dao.StudentInsertDAOImpl">
<property name="jt" ref="template"/>
</bean>
</beans>
```

student.jsp:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<h1><center> Register Student Details</center></h1>
<form:form method="POST" commandName="stCmd">
  Number:  <form:input path="sno"/><br>
  Name  : <form:input path="sname"/><br>
  Address : <form:input path="sadd"/><br>
<input type="submit" value="register"/>
</form:form>
```

result.jsp:

```
<h1><center> Result Page</center></h1>
${resMsg}
```

Form validation in SimpleformController:

- Verifying the format and pattern of form data is called form validation. Spring MVC allows to write form validation logic in separate class so that it can be linked with multiple controller classes having reusability.

Sample Code:

- Develop a validator class implementing **validation(i)** and implement**supports(-)** and **validate (-,-)** methods.
- Develop properties file having validation Error messages.
- Configure the properties file in "dispatcher-servlet.xml"
- Configure validator class in dispatcher-Servlet.xml and link that class with controller class using the property **"validator"**
- Place <form:errors path="*"/>tag in student.jsp to display the form validation error messages.

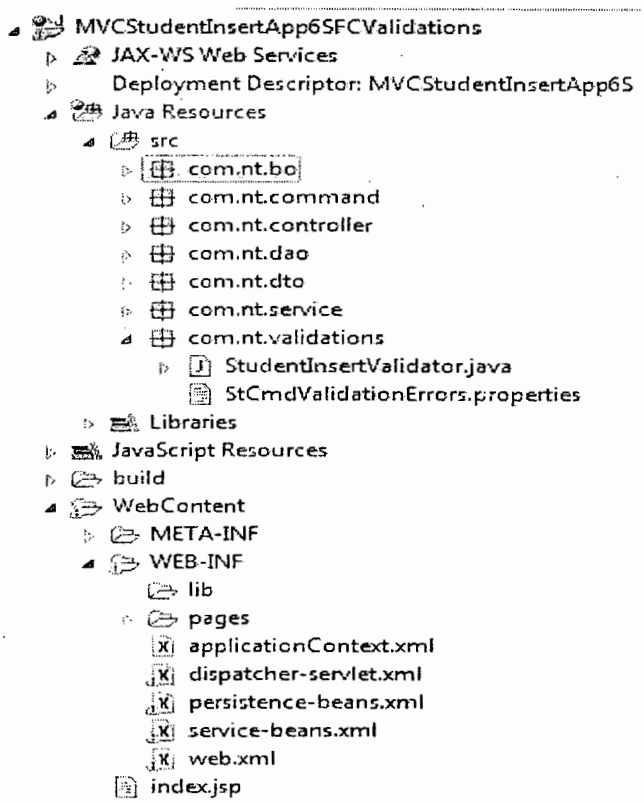
The SimpleFormController can handle 3 types of validation:

a) **Form Validations** (like number, name can't be blank).
b) ***typeMismatch*** errors (like storing stirng value to Sno properties).
c) **Application logic errors** (like not allowing "delhi" as address for student) Instead of hardcoading error msgs directly in validator class and controller class it is recommended to place in properties file having our choice keys and values, but **typemismatch errors** key should be **"typeMismatch.<propertyname>"**

Q) What is the use of implementing supports (Class clazz) method in validator class of SimpleFormController?

=>This supports (-) method holds the configured "CommandClass"as"Class" parameter and we can check wheather correct command class is configured or not by using **Class.isAssignableFrom(-)** method, if we configure correctly then it returns true, so validate (-,-) method executes, if not configured correctly then it returns false, so validate(-,-) will not execute. Due to this we can avoid mis configuration in spring bean configure file.

```
org.springframework.validation.Errors(i)
                ^
                |  Extends
                |

org.springframework.validation.BindingResults(i)
                ^      implements
                        |
                        |
org.springframework.validation.BindException(c)
```



StudentBO.java:

```
package com.nt.bo;
public class StudentBO {
    private int sno;
    private String sname,sadd;
    public int getSno() {
            return sno;
    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
            this.sadd = sadd;
    }
}
```

StudentInsertDAO.java:

```
package com.nt.dao;
import com.nt.bo.StudentBO;
public interface StudentInsertDAO {
    public int insert(StudentBO sbo);
}
```

StudentInsertDAOImpl.java:

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
import com.nt.bo.StudentBO;
public class StudentInsertDAOImpl implements StudentInsertDAO {
    private static final String  INSERT_STUDENT_QRY="INSERT INTO STUDENT
values(?,?,?)";
    private JdbcTemplate jt;
    public void setJt(JdbcTemplate jt) {
            this.jt = jt;
    }
    @Override
    public int insert(StudentBO sbo) {
    int cnt=jt.update(INSERT_STUDENT_QRY,sbo.getSno(),
                    sbo.getSname(),
                    sbo.getSadd());
```

```
    return cnt;
    }
    }
```

StudentInsertService.java:

```
package com.nt.service;
import com.nt.dto.StudentDTO;
public interface StudentInsertService {
    public String  register(StudentDTO dto);
}
```

StudentDTO.java:

```
package com.nt.dto;
import java.io.Serializable;
public class StudentDTO implements Serializable {
    private int sno;
    private String sname,sadd;
    public int getSno() {
            return sno;
    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
            this.sadd = sadd;
    }
}
```

StudentInsertServiceImpl.java:

```
package com.nt.service;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentInsertDAO;
import com.nt.dto.StudentDTO;
public class StudentInsertServiceImpl implements StudentInsertService{
    private StudentInsertDAO dao;
public void setDao(StudentInsertDAO dao) {

            this.dao = dao;
    }
    @Override
```

```
    public String register(StudentDTO dto) {
        // Convert DTO object BO object
        StudentBO sbo=new StudentBO();
        sbo.setSno(dto.getSno());
        sbo.setSname(dto.getSname());
        sbo.setSadd(dto.getSadd());
        //Use DAO
        int cnt=dao.insert(sbo);
        if(cnt==0)
            return " Student Regisration Failed";
        else
            return " Student Regisration Succeded with no"+sbo.getSno();
    }}
```

StudentCommand.java:

```
package com.nt.command;
public class StudentCommand {
    private int sno=101;
    private String sname="raja",sadd="hyd";
    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
}
```

StCmdValidationErrors.properties

```
typeMismatch.sno=student no can  not empty or can not be String
sno.invalid=Student no must be positive value
sname.required=Student name is mandatory
sadd.required=Student address is mandatory
sadd.minlength= Addrs can not be less than 3 chars
sadd.city=Address can be not  "delhi".
```

StudentInsertValidator.java:

```
package com.nt.validator;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.nt.command.StudentCommand;
public class StudentInsertValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {

            return clazz.isAssignableFrom(StudentCommand.class);
 }
 @Override
 public void validate(Object command, Errors errors) {
    //type casting
    StudentCommand cmd=(StudentCommand)command;
    //form validations
    if(cmd.getSno()<=0)
      errors.rejectValue("sno","sno.invalid");
    if(cmd.getSname().equals("") || cmd.getSname()==null)
            errors.rejectValue("sname", "sname.required");
    if(cmd.getSadd().equals("") || cmd.getSadd()==null)
            errors.rejectValue("sadd", "sadd.required");
    else{
            if(cmd.getSadd().length()<3){
                    errors.rejectValue("sadd","sadd.minlength");
            }
    }
            }
}
```

StudentInsertController.java:

```
package com.nt.controller;
import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.nt.command.StudentCommand;
import com.nt.dto.StudentDTO;
import com.nt.service.StudentInsertService;
public class StudentInsertController extends SimpleFormController {
  private StudentInsertService  service;
 public void setService(StudentInsertService service) {
    this.service = service;
 }
```

```
@Override
 protected ModelAndView handleInvalidSubmit(HttpServletRequest req,
            HttpServletResponse res) throws Exception {
    return new ModelAndView("dblpost");
 }
 @Override
 protected ModelAndView onSubmit(HttpServletRequest request,
            HttpServletResponse response, Object command, BindException errors)
            throws Exception {
    //type cast cmd object
    StudentCommand cmd=(StudentCommand)command;
            // B.logic related validation
            if(cmd.getSadd().equals("delhi")){
                    errors.rejectValue("sadd","sadd.city");
                    return showForm(request,response, errors) ;
            }
                        //Convert Command object to DTO object
            StudentDTO  dto=new StudentDTO();
            dto.setSno(cmd.getSno());
            dto.setSname(cmd.getSname());
            dto.setSadd(cmd.getSadd());
            //Use Service class
            String result=service.register(dto);
            //return ModelAndView object
            return new ModelAndView(this.getSuccessView(),"resMsg",result);

    }
            @Override
    protected Object formBackingObject(HttpServletRequest request)
                    throws Exception {

            int no=Integer.parseInt(request.getParameter("id"));
      StudentCommand cmd=new StudentCommand();
      cmd.setSno(no);
      cmd.setSname("raja111");
      cmd.setSadd("hyd111");
      return cmd;
    }
}
```

StudentCommand.java:

```
package com.nt.command;
public class StudentCommand {
    private int sno=101;
    private String sname="raja",sadd="hyd";
    public int getSno() {
            return sno;
    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
            this.sadd = sadd;
    }
}
```

index.jsp:

```
<jsp:forward page="student.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/student.htm">sic</prop>
</props>
</property>
</bean>
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
<property name="basename" value="com/nt/validator/StCmdValidationErrors"/>
</bean>
<bean id="stValidator"  class="com.nt.validator.StudentInsertValidator"/>
<bean id="sic"  class="com.nt.controller.StudentInsertController">
<property name="formView" value="student"/>
<property name="validator" ref="stValidator"/>
<property name="successView" value="Result"/>
<property name="commandClass" value="com.nt.command.StudentCommand"/>
<property name="commandName" value="stCmd"/>
<property name="sessionForm" value="true"/>
<property name="service"  ref="stService"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<import resource="persistence-beans.xml"/>
<import resource="service-beans.xml"/>
</beans>
```

service-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="stService"  class="com.nt.service.StudentInsertServiceImpl">
<property name="dao"  ref="stDao"/>
</bean>
</beans>
```

persistence-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<bean id="stDao"  class="com.nt.dao.StudentInsertDAOImpl">
<property name="jt" ref="template"/>
</bean>
</beans>
```

student.jsp:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<h1><center> Register Student Details</center></h1>
<form:form method="POST" commandName="stCmd">
<!-- <font color='red'> -->
<%--   <form:errors path="*"/>  --%>
<!-- </font>   -->
<!-- <br> -->
 Number:  <form:input path="sno"/><form:errors path="sno"/><br>
 Name  : <form:input path="sname"/><form:errors path="sname"/><br>
 Address : <form:input path="sadd"/><form:errors path="sadd"/><br>
<input type="submit" value="register"/>
</form:form>
```

result.jsp:

```
<h1><center> Result Page</center></h1>
${resMsg}
```

dbpost.jsp:

```
<h1> Warning!!!!</h1>
<p style="color:red"> double posting is not allowed </p>
```

Internal flow of form validation:

- Form page Submits the form (post back request)----->Dispatcher servlet traps request and uses handler mapping to map the request to our SimpleFormController and calls handleRequest (-,-)--------->this method checks wheather request is "initialphase" or "postback" request. Since the request is post back request.

a) Creates an empty BindException object
b) Creates or locates command class object and writes form data to it.
(request wrapping)
c) Checks "validation" is configured or not and it notices configured in controller class.
d) get validator object and calls supports(-) with Class object that points command Class if this method returns true then validate(-,-) will be called having Errors Object (BindException), command object otherwise validate(-,-) will not be called.
e) validate(-,-) perform form validations and stores validationErrors in Errors object.
f) handleRequest(-,-) checks the size of Error object to know weather validation Errors are there or not there.

If Errors object size=0 (Then no validation errors)
Then {
Calls onSubmit(-,-,-,-) having req,res,command and BindException objects.
onSubmit(-,-,-,-) will process the request
}
If (Error object size>0) [form validation errors]
{

- keep Error objects in request scope.
- Returns MAV object having formview property value as lvn, commandClassName, CommandClas object as model attribute.
- DispatcherServlet keeps the received command class object in req scope and use ViewResolver to get view object having form page as physical view Name.
- Form page will be displayed. In that <form:input> tags collect command object data to show old values in comps and <form: errors> uses "Error object" data to show validation Error messages.

}

Q)What is double posting problem and duplicate form submission problem and how can we solve that problem?

- Pressing refresh button on the result page of form submission submits the form one more time, clicking on submit button for multiple times continuously also raises this problem. This leads duplicate form submission and gives problem like duplicate

registrations, deducting amount from credit card,debit card accounts for multiple times and etc..... .This is also called as "doublePosting" problem.

- We can overcome this problem with tokens support, when form page is submitted one token will be created and will be maintained as session attribute. Once onSubmit(-,-) method process the request it removes the token from session. If any request comes without token because of double posting problem error will be raised.

- In spring MVC all these activities are handle internally if any request comes without token, the handleInvalidSubmit (-,-) method will be called. So we can place logic to render error page in that method.

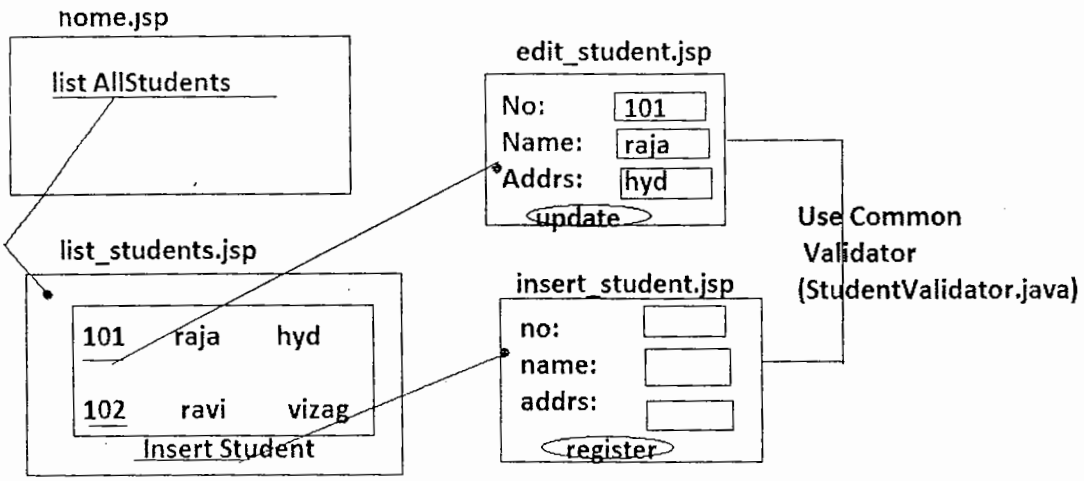
Importance of formBackingObject ()

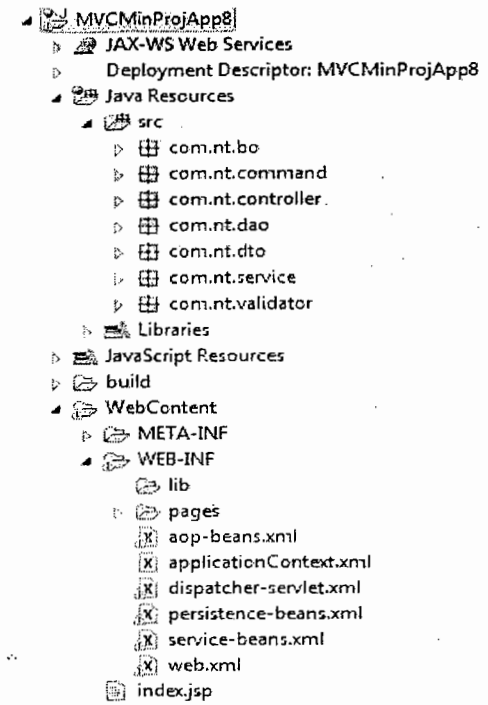
- Generally when handleRequest method is processing initial phase request it internally calls formBackingObject() to create command class object. The predefined method always creates empty command object. If you want to create command object with dynamic data according to application requirement then we need to override this method in our controller class that extends from SimpleformController.

Usecase:

- When we clicked on student no hyperlink from list of student details, if we want to display edit.jsp having that student details initial as values.

Diagram:





StudentBO.java:

```
package com.nt.bo;
public class StudentBO {
    private  int sno;
    private String sname;
    private String sadd;
    public int getSno() {
            return sno;
    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
this.sadd = sadd;  }  }
```

StudentDAO.java:

```
package com.nt.dao;
import java.util.List;
import com.nt.bo.StudentBO;
public interface StudentDAO {
    public  List<StudentBO> getAllStudents();
    public  int insert(StudentBO bo);
    public  StudentBO  getStudent(int no);
    public int update(StudentBO bo);
}
```

StudentDAOImpl.java:

```
package com.nt.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import com.nt.bo.StudentBO;
public class StudentDAOImpl implements StudentDAO {
     private static final String  GET_ALL_STUDENTS_QRY="SELECT SNO,SNAME,SADD
FROM      STUDENT ";
    private static final String  INSERT_STUDENT_QRY="INSERT INTO STUDENT
VALUES(?,?,?)";
     private static final String  GET_STUDENT_BY_SNO="SELECT SNO,SNAME,SADD FROM
    STUDENT WHERE SNO=?";
     private static  final  String UPDATE_STUDENT_BY_NO="UPDATE STUDENT SET
SNAME=? ,        SADD=? WHERE SNO=?";

     private JdbcTemplate  jt;
      public void setJt(JdbcTemplate jt) {
            this.jt = jt;
    }
    @Override
    public List<StudentBO> getAllStudents() {
            List<StudentBO> listBO=jt.query(GET_ALL_STUDENTS_QRY, new
    StudentRowMapper());
            return listBO;
    }
    private class StudentRowMapper implements RowMapper<StudentBO>{
    @Override
            public StudentBO mapRow(ResultSet rs, int pos) throws SQLException {
                    StudentBO bo=new StudentBO();
                    bo.setSno(rs.getInt(1));
                    bo.setSname(rs.getString(2));
                    bo.setSadd(rs.getString(3));
                  return bo;
            }
```

```
        }//inner class
    @Override
    public int insert(StudentBO bo) {
        int cnt=jt.update(INSERT_STUDENT_QRY, bo.getSno(),
bo.getSname(),
bo.getSadd());
        return cnt;
    }
    @Override
    public StudentBO getStudent(int no) {
        StudentBO bo=jt.queryForObject(GET_STUDENT_BY_SNO, new
    StudentRowMapper(), no);
        return bo;
    }
    public int update(StudentBO bo){
        int cnt=jt.update(UPDATE_STUDENT_BY_NO,
    bo.getSname(),bo.getSadd(),bo.getSno());
        return cnt;
    }
}
```

StudentDTO.java:

```
package com.nt.dto;
import java.io.Serializable;
public class StudentDTO implements Serializable {
    private  int sno;
    private String sname;
    private String sadd;
    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
}
```

StudentService.java:

```
package com.nt.service;
import java.util.List;
import com.nt.dto.StudentDTO;
public interface StudentService {
    public  List<StudentDTO> listStudents();
    public  String regsiterStudent(StudentDTO dto);
    public  StudentDTO  searchStudentBySno(int no);
    public String updateStudentBySno(StudentDTO dto);
}
```

StudentServiceImpl.java:

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import com.nt.bo.StudentBO;
import com.nt.dao.StudentDAO;
import com.nt.dto.StudentDTO;
public class StudentServiceImpl implements StudentService{
    private StudentDAO dao;
    public void setDao(StudentDAO dao) {
            this.dao = dao;
    }
    @Override
    public List<StudentDTO> listStudents() {
            //use dao
            List <StudentBO> listBO=dao.getAllStudents();
            //Copy ListBO to List DTO
            List<StudentDTO> listDTO=new ArrayList<StudentDTO>();
            if(listBO!=null){
                    for(StudentBO  bo:listBO){
                            StudentDTO dto=new StudentDTO();
                            dto.setSno(bo.getSno());
                            dto.setSname(bo.getSname());
                            dto.setSadd(bo.getSadd());
                            listDTO.add(dto);
                    }//for
            }//for
            return listDTO;
    }//method
@Override
    public String regsiterStudent(StudentDTO dto) {
            //Convert DTO object BO object
            StudentBO bo=new StudentBO();
            bo.setSno(dto.getSno());
            bo.setSname(dto.getSname());
            bo.setSadd(dto.getSadd());
            //use dao.

            int cnt=dao.insert(bo);
```

```
ListBo.forEach(bo-->{
StudentDTO dto=null;
dto=new StudentDTO();
BeanUtils.copyProperties(bo.dto);
ListDTO.add(dto);
});
return ListDTO;
```

```
        if(cnt==0)
                return "Student Registration Failed";
        else
                return "Student Registration Succeded with no"+bo.getSno();
    }//registerStudent
    @Override
    public StudentDTO searchStudentBySno(int no) {
        //use DAO
        StudentBO bo=dao.getStudent(no);
        //convert BO object to DTO object
        StudentDTO dto=new StudentDTO();
        dto.setSno(bo.getSno());
        dto.setSname(bo.getSname());
        dto.setSadd(bo.getSadd());
        return dto;
    }
    @Override
    public String updateStudentBySno(StudentDTO dto) {
      //convert DTO object to BO object
        StudentBO bo=new StudentBO();
        bo.setSno(dto.getSno());
        bo.setSname(dto.getSname());
        bo.setSadd(dto.getSadd());
        //use DAO
        int cnt=dao.update(bo);
        if(cnt==0)
                return dto.getSno()+" student not for updation";
        else
                return dto.getSno()+" student details are updated";
    }
}//class
```

StCmdValidationErrors.properties:

```
typeMismatch.sno=student no can  not empty or can not be String
sno.invalid=Student no is mandatory and must be positive
sname.required=Student name is mandatory
sadd.required=Student address is mandatory
sadd.minlength= Addrs can not be less than 3 chars
sadd.city=Address can be not  "delhi".
```

StudentInsertValidator.java:

```
package com.nt.validator;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.nt.command.StudentCommand;
public class StudentInsertValidator implements Validator {
    @Override
    public boolean supports(Class<?> clazz) {
        return clazz.isAssignableFrom(StudentCommand.class);
```

```
    }
    @Override
    public void validate(Object command, Errors errors) {
            //type casting
            StudentCommand cmd=(StudentCommand)command;
            //form validations
            if(cmd.getSno()==null || cmd.getSno()<=0)
              errors.rejectValue("sno","sno.invalid");
            if(cmd.getSname().equals("") || cmd.getSname()==null)
                    errors.rejectValue("sname", "sname.required");
            if(cmd.getSadd().equals("") || cmd.getSadd()==null)
                    errors.rejectValue("sadd", "sadd.required");
            else{
                    if(cmd.getSadd().length()<3){
                            errors.rejectValue("sadd","sadd.minlength");
                    }
            }
    }
}
```

ListStudentsController.java:

```
package com.nt.controller;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import com.nt.dto.StudentDTO;
import com.nt.service.StudentService;
public class ListStudentsController extends AbstractController {
    private StudentService service;
    public void setService(StudentService service) {
            this.service = service;
    }
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
            //use SErivce class
```

```
    List<StudentDTO> listDTO=service.listStudents();
            //create MAV object
            ModelAndView mav=new ModelAndView();
            mav.setViewName("list_students");
            mav.addObject("stList",listDTO);
            return mav;
    }
}
```

StudentInsertController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.nt.command.StudentCommand;
import com.nt.dto.StudentDTO;
import com.nt.service.StudentService;
public class StudentInsertController extends SimpleFormController {
    private StudentService service;

    public void setService(StudentService service) {
            this.service = service;
    }
            @Override
    protected ModelAndView onSubmit(HttpServletRequest req,
                    HttpServletResponse res, Object command, BindException errors)
                    throws Exception {
            // type cast command
            StudentCommand cmd=(StudentCommand)command;
            //convert Command object to DTO object
            StudentDTO  dto=new StudentDTO();
            dto.setSno(cmd.getSno());
            dto.setSname(cmd.getSname());
            dto.setSadd(cmd.getSadd());
            //use service
            String result=service.regsiterStudent(dto);
            return new ModelAndView("insert_result","resMsg",result);
    }
    @Override
    protected ModelAndView handleInvalidSubmit(HttpServletRequest request,
                    HttpServletResponse response) throws Exception {
            return new ModelAndView("dblpost");
            }}
```

StudentUpdateController.java

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import com.nt.command.StudentCommand;
import com.nt.dto.StudentDTO;
import com.nt.service.StudentService;
public class StudentUpdateController extends SimpleFormController {
```

```
    private StudentService service;

    public void setService(StudentService service) {
            this.service = service;
    }
    @Override
    protected Object formBackingObject(HttpServletRequest request)
                    throws Exception {
            //read sno
            int no=Integer.parseInt(request.getParameter("id"));
            //use Service class
            StudentDTO dto=service.searchStudentBySno(no);
            //create Command class object and place DTO object
            StudentCommand  cmd=new StudentCommand();
            cmd.setSno(dto.getSno());
            cmd.setSname(dto.getSname());
            cmd.setSadd(dto.getSadd());
            return cmd;
    }//method
    @Override
    protected ModelAndView onSubmit(HttpServletRequest request,
                    HttpServletResponse response, Object command, BindException errors)
                    throws Exception {
            // typecast command
            StudentCommand cmd=(StudentCommand)command;
            //convert Command object to DTO object
            StudentDTO dto=new StudentDTO();
            dto.setSno(cmd.getSno());
            dto.setSname(cmd.getSname());
            dto.setSadd(cmd.getSadd());
            //use Service class
            String updMsg=service.updateStudentBySno(dto);
            return new ModelAndView(getSuccessView(),"updResult",updMsg);
    }
}
```

index.jsp:

```
<jsp:forward page="home.htm"/>
```

web.xml:

```
<web-app>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

```
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
    <bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
    <props>
    <prop key="/home.htm">pvc1</prop>
    <prop key="/listStudents.htm">stLc</prop>
    <prop key="/insert.htm">sic</prop>
    <prop key="/edit.htm">suc</prop>
    </props>
    </property>
    </bean>
    <bean id="pvc1"
        class="org.springframework.web.servlet.mvc.ParameterizableViewController">
    <property name="viewName"  value="home"/>
    </bean>
    <bean id="stLc"  class="com.nt.controller.ListStudentsController">
    <property name="service"  ref="stService"/>
    </bean>
    <bean id="messageSource"
        class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="com/nt/validator/StCmdValidationErrors"/>
    </bean>
    <bean id="stValidator"  class="com.nt.validator.StudentInsertValidator"/>
    <bean id="sic"  class="com.nt.controller.StudentInsertController">
    <property name="formView" value="insert_student"/>
    <property name="successView" value="insert_result"/>
    <property name="commandClass" value="com.nt.command.StudentCommand"/>
    <property name="commandName" value="stCmd"/>
    <property name="validator"  ref="stValidator"/>
    <property name="sessionForm" value="true"/>
    <property name="service"  ref="stService"/>
```

```
</bean>
<bean id="suc" class="com.nt.controller.StudentUpdateController">
<property name="formView" value="edit_student"/>
<property name="successView" value="edit_result"/>
<property name="commandClass" value="com.nt.command.StudentCommand"/>
<property name="commandName" value="stCmd"/>
<property name="validator" ref="stValidator"/>
<property name="sessionForm" value="true"/>
<property name="service" ref="stService"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/pages/"/>
<property name="suffix" value=".jsp"/>
</bean>
</beans>
```

applcationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<import resource="persistence-beans.xml"/>
<import resource="service-beans.xml"/>
<import resource="aop-beans.xml"/>
</beans>
```

persistence-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="jofb" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName" value="DsJndi"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="jofb"/>
</bean>
<bean id="stDao" class="com.nt.dao.StudentDAOImpl">
<property name="jt" ref="template"/>
</bean>
</beans>
```

service-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="stService"  class="com.nt.service.StudentServiceImpl">
<property name="dao"  ref="stDao"/>
</bean>
</beans>
```

aop-beans.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop.xsd">
  <bean id="txMgr"
     class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource"  ref="jofb"/>
  </bean>
  <tx:advice id="txAdvice" transaction-manager="txMgr">
  <tx:attributes>
  <tx:method name="list*"  propagation="REQUIRED"
                                                read-only="true"/>
  <tx:method name="register*"  propagation="REQUIRED"
                                                read-only="false"/>
  <tx:method name="search*"  propagation="REQUIRED"
                                                read-only="true"/>
  <tx:method name="update*"  propagation="REQUIRED"
                                                read-only="false"/>
  </tx:attributes>
  </tx:advice>
  <aop:config>
  <aop:pointcut id="ptc1"
                    expression="execution(* com.nt.service.StudentService.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="ptc1"/>
  </aop:config>
  </beans>
```

dbpost.jsp:

```
<h1 style="color:red"><center> Double Posting is not Alloed </center></h1>
<a href="home.htm">home</a>
```

edit_result.jsp:

```
<h1><center> Result Page</center></h1>
 ${updResult}
<a href="home.htm">home</a>
```

edit_student.jsp:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<h1><center> Update Student Details</center></h1>
<form:form  method="post"  commandName="stCmd">
   Sno : <form:input path="sno"/><span style="color:red"><form:errors
path="sno"/></span>        <br>
   Sname : <form:input path="sname"/><span style="color:red"><form:errors
path="sname"/>      </span><br>
   Sadd : <form:input path="sadd"/><span style="color:red"><form:errors
path="sadd"/>        </span><br>
<input type="submit"  value="update"/>
</form:form>
```

home.jsp:

```
<h1> Home Page</h1>
<a href="listStudents.htm">List All Students</a>
```

insert_result.jsp:

```
<h1><center> Result page</centeR></h1>
${resMsg}
<br>
<a href="home.htm">home</a>
```

insert_student.jsp:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<h1><center> Register Student Details</center></h1>
<form:form  method="post"  commandName="stCmd">
   Sno : <form:input path="sno"/><span style="color:red"><form:errors
path="sno"/></span>        <br>
   Sname : <form:input path="sname"/><span style="color:red"><form:errors
path="sname"/>      </span><br>
   Sadd : <form:input path="369ad"/><span style="color:red"><form:errors
path="369ad"/>      </span><br>
<input type="submit"  value="register"/>
</form:form>
```

List_students.jsp:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<h1><center>  Student Details</center></h1>
<table bgcolor=red border=1 width="100%" >
<tr bgcolor="green"><th>Sno </th><th>Sname</th><th>sadd</th></tr>
<c:choose>
<c:when test="${stList ne null }">
<c:forEach var="st" items="${stList}">
<tr>
<td><a href="edit.htm?id=${st.sno}"><c:out value="${st.sno }" /></a></td>
<td><c:out value="${st.sname }" /></td>
<td><c:out value="${st.sadd }" /></td>
</tr>
</c:forEach>
</c:when>
<c:otherwise>
<span style="color:red"> No Records Found </span>
</c:otherwise>
</c:choose>
</table>
<br><br>
<center>
<a href="insert.htm"> Insert Student</a>
</center>
<br>
<br>
<a href="home.htm">home</a>
```

referenceData (HttpServletRequest)

- While displaying form page as the response of initial phase request or as the response of post back request when form validation errors are there.if you want to display additional data apart from the data that is there in command object, we can override referenceData(-) in our SimpleFormController class to construct that data dynamically. This method returns Map object having data in the form of values List with key.
- DispatcherServlet keeps this Map object in request scope. In order to display this Map object data in certain component we need the key of Map object in the "items" attribute of <form:options>,<form:checkboxes>and etc tags...

Method signature:

```
Public Map referenceData(HttpServletRequest request)
```

Understanding how various methods will be called internally from handleRequest() of SimpleFormController:

1.InitialRequest
Form Display
GET Request

Call sequence
1.FormBacking Object(...)
2.InitBinder(...)
3.ShowForm(...)
4.referenceData(...)

=>

2.InvalidForm Submission
POST Request

Call Sequence
1.initBinder(...)
2.onBind(...)
3.onBindAndValidator(...)
4.ProcessFormSubmission(...)
5.shoeForm(...)
6.referenceData(...)

=>

3.ValiForm Submission
POST Request

Call Sequence
1.initBinder(...)
2.onBind(...)
3.onBindAndValidator(...)
4.ProcessFormSubmission(...)
5.onSubmit(...)
6.redirect to a GET

Request1(form GET)
user first enter the screen1
after this request the form that is displayed in the user's browser.

request2(failedSubmission)
User submit the form with invalid fields.
(validiation fails)

Request3(Sucessful Submission)
Show which method are called on successful submission

Note:
1-show form method collects and returns formview property value as logical view name
2-onbind method reads the form data and stores into command object(request wrapping)
3-onbindedvalidate method forms forms validations and stores errors in bind exception object
4-processformsubmisstion method takes bindexception object and checks whether form validation errors there are not there if there it calls showform method otherwise it calls onsubmit method

The spring tags generated radio buttons, List boxes, Select box and Checkboxes can't display initial/old values collected from command object for this we need to construct that additional/old values for these components by overriding ***referenceData(-)***.

- MVCStudentApp7SPRINGFRAMEWORKC
  - JAX-WS Web Services
  - Deployment Descriptor: MVCStudentApp7SPRIN(
  - Java Resources
    - src
      - com.nt.command
        - RegistrationCommand.java
      - com.nt.controller
        - StudentRegistrationController.java
      - com.nt.validations
        - StudentInsertValidator.java
        - StCmdValidationErrors.properties
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
      - pages
      - applicationContext.xml
      - dispatcher-servlet.xml
      - persistence-beans.xml
      - service-beans.xml
      - web.xml
    - index.jsp

Note: all the methods given in above three boxes are actually called from handlerequest method of SimpleFormController

Add Jars: Spring 3.2.6 jar files

StudentRegistrationController.java:

```
package com.nt.controller;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
public class StudentRegistrationController extends SimpleFormController {
        @Override
    protected Map referenceData(HttpServletRequest request) throws Exception {
        System.out.println("SRC: referenceData(-)");
        Map<String,List> map=new HashMap<String,List>();
          List <String> courses=new ArrayList();
          courses.add("spring"); courses.add("hibernate");
          courses.add("adv.java"); courses.add("webservices");
         map.put("courseList",courses);
         return map;
        }
    @Override
    protected ModelAndView onSubmit(HttpServletRequest request,
                HttpServletResponse response, Object command, BindException errors)
                throws Exception {
        return new ModelAndView("register_result");
    }
        }
```

RegistrationCommand.java:

```
package com.nt.command;
public class RegistrationCommand {
    private int sno;
    private String sname;
    private String course;
    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
```

```
    }
    public String getCourse() {
        return course;
    }
    public void setCourse(String course) {
        this.course = course;
    }
}
```

StCmdValidationErrors.java:

```
typeMismatch.sno=student no can  not empty or can not be String
sno.invalid=Student no must be positive value
sname.required=Student name is mandatory
```

StudentInsertValidator.java

```
package com.nt.validator;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.nt.command.RegistrationCommand;
public class StudentInsertValidator implements Validator {
    @Override
    public boolean supports(Class<?> clazz) {
        return clazz.isAssignableFrom(RegistrationCommand.class);
    }
    @Override
    public void validate(Object command, Errors errors) {
        //type casting
        RegistrationCommand cmd=(RegistrationCommand)command;
        //form validations
        if(cmd.getSno()<=0)
          errors.rejectValue("sno","sno.invalid");
        if(cmd.getSname().equals("") || cmd.getSname()==null)
            errors.rejectValue("sname", "sname.required");
        }
}
```

index.jsp:

```
<jsp:forward page="register.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

dispatcher-servlet.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-3.0.xsd">
<bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/register.htm">src</prop>
</props>
</property>
</bean>
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
<property name="basename" value="com/nt/validator/StCmdValidationErrors"/>
</bean>
<bean id="stValidator"  class="com.nt.validator.StudentInsertValidator"/>
<bean id="src"  class="com.nt.controller.StudentRegistrationController">
<property name="formView" value="register"/>
<property name="validator"  ref="stValidator"/>
```

```
<property name="successView" value="register_result"/>
<form:options items="${courseList}"/><br>
</form:select>
<input type="submit" value="register"/>
</form:form>
<property name="commandClass" value="com.nt.command.RegistrationCommand"/>
<property name="commandName" value="regCmd"/>
<property name="sessionForm" value="true"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

result.jsp:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<form:form method="POST" commandName="regCmd">
<form:errors path="*"/>
<br>
 No : <form:input  path="sno"/><br>
 Name : <form:input path="sname"/><br>
 Courses :
<form:select path="course">
```

register_result.jsp

```
<b><center> Registration Successspringframeworkul</center></b>
```

Usecase:

➔Displaying roles in formpage (Select box)

➔For initial phase request and it in post back request when it generates form validation errors.

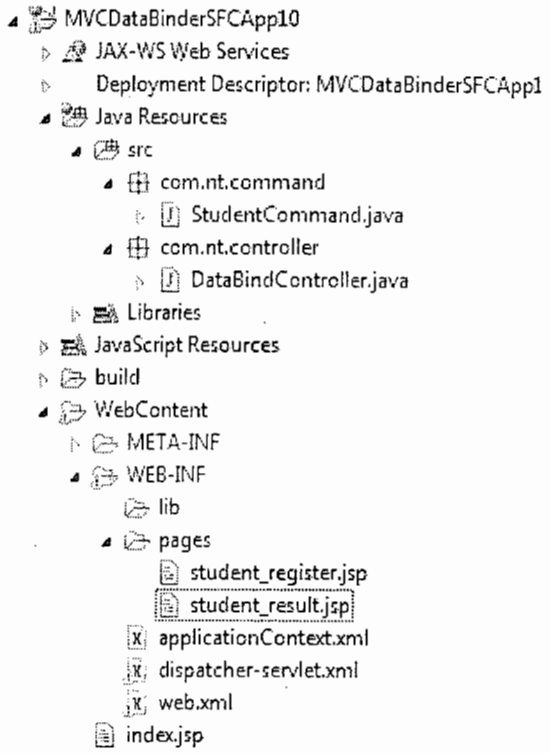
initBinder(-,-) :-

- ⇨ property editors of spring are very useful to convert given values to required type properties of Bean class or command class.
- ⇨ Spring gives built-in property editors and allows us to develop custom property editors. **CustomDateEditor** is given to convert spring data values into java.util.Date class object by specifying date formate.
- ⇨ While working with SimpleFormController we can override initBinder(-,-) method to register propertyeditors that are required to use while binding given form data (request param values) to command class object properties.
- ⇨ This method gives request, ServletRequestDataBinder objects. We can register property editors with servletRequestDataBinder object because it is responsible to bind the form submitted request parameter values to command object properties.

Use case:

Storing string value of form component to command class property as java.util.Date class object.

Internal flow of execution in handleRequest method.



Add jars: Spring 3.2.6 jars

StudentCommand.java:

```
package com.nt.command;
import java.util.Date;
public class StudentCommand {
    private String name;
    private Date  dob,doj,dom;
    public String getName() {
            return name;
    }
    public void setName(String name) {
            this.name = name;
    }
    public Date getDob() {
            return dob;
    }
    public void setDob(Date dob) {
            this.dob = dob;
    }
    public Date getDoj() {
            return doj;
    }
    public void setDoj(Date doj) {
            this.doj = doj;
    }
    public Date getDom() {
            return dom;
    }
```

```
        public void setDom(Date dom) {
                this.dom = dom;
        }
        @Override
        public String toString() {
                return "StudentCommand [name=" + name + ", dob=" + dob + ", doj=" + doj
                        + ", dom=" + dom + "]";
        }
    }
```

DataBindController.java:

```
    package com.nt.controller;
    import java.text.SimpleDateFormat;
    import java.util.Date;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import org.springframework.beans.propertyeditors.CustomDateEditor;
    import org.springframework.validation.BindException;
    import org.springframework.web.bind.ServletRequestDataBinder;
    import org.springframework.web.servlet.ModelAndView;
    import org.springframework.web.servlet.mvc.SimpleFormController;
    import com.nt.command.StudentCommand;
    public class DataBindController extends SimpleFormController {
        @Override
        protected void initBinder(HttpServletRequest request,
                        ServletRequestDataBinder binder) throws Exception {
                SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
                binder.registerCustomEditor(Date.class,
                                                        new
CustomDateEditor(sdf,true));
        }
                @Override
        protected ModelAndView onSubmit(HttpServletRequest request,
                        HttpServletResponse response, Object command, BindException errors)
                        throws Exception {
                StudentCommand cmd=(StudentCommand)command;
                return new ModelAndView("student_result","cmdData",cmd);
        }
}
```

Index.jsp

```
<jsp:forward page="student.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/student.htm">dbc</prop>
</props>
</property>
</bean>
<bean id="dbc"  class="com.nt.controller.DataBindController">
<property name="formView" value="student_register"/>
<property name="successView" value="student_result"/>
<property name="commandClass" value="com.nt.command.StudentCommand"/>
<property name="commandName" value="regCmd"/>
<property name="sessionForm" value="true"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

appliactionContext.xml:

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

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

student_register.jsp:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<form:form method="POST" commandName="regCmd">
<form:errors path="*"/>
  Name :  <form:input path="name"/><br>
  DOB   : <form:input path="dob"/><br>
  DOJ   : <form:input path="doj"/><br>
  DOM   : <form:input path="dom"/><br>
<input type="submit" value="register"/>
</form:form>
```

student_result.jsp:

```
The given details are :
  ${cmdData}
```

MultiActionController:

- ⇨ When form page contains multiple submit buttons, then instead of taking multiple AbstractCommandController/SimpleFormController classestake single MultiActionController having multiple user-defined methods to process request of multiple submit buttons on one per method basis.
- ⇨ The userdefined methods of this class can have following standard signature

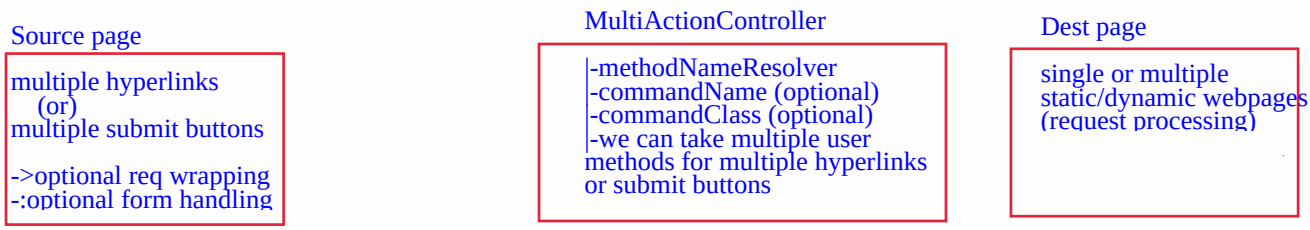
```
public (C \Map\String\void) <methodname>
(HttpServletRequest request, HttpServletResponse response, [HttpSession],
[anyObject]);
```

a) **ModelAndView**(lvn+model)
b) **Map**(only Model data), So **virtual path** of request uri becomes **view name**).
c) **void** (no Model data , So **virtual path** becomes **view name**).
d) **String**(no model data, **only logical view name**).

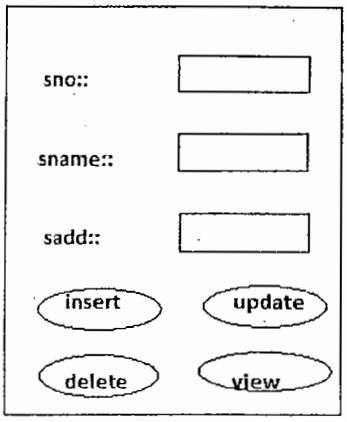
While configuring our MultiActionController class we must specify the MethodNameResolver object as dependent object that contains additional parameter name based on which the method that has to be executed in MultiActionController class will be decided.

Conclusion

- ⇨ If source page is form submission having multiple submit buttons with optional form validation and the required destination page is static/dynamic webpage that should come after request processing then we should go after MultiActionController.



eg:

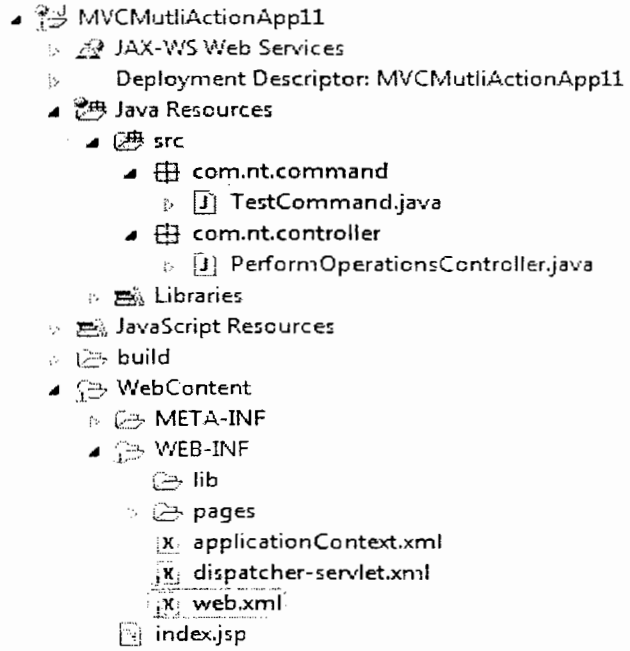



```
public PerformOperationsController extends MultiActionController
{
   public MAV add(-,-,-){
      ------
      ------
}

 public MAV update(-,-,-){
      ------
      ------
}

 public MAV delete(-,-,-){
      ------
      ------
}

 public MAV view(-,-,-){
      ------
      ------
}
```

Flow for insert submit button:

- ⇨ Insert submit button---->DispatcherServlet----->HandlerMapping----->controller (MAController)------>handleRequest(-,-)------->create command object and performs request wrapping.
- ⇨ Reads paramName(opt), defaultMethodName(invalid) from the injected MethodNameresolver object.

Reads the values of "opt" req param (insert)----->Calls insert (req,res, command) method.



Add Jars: Spring 4.x jar of MVC

eg:

TestCommand.java:

```
package com.nt.command;
public class TestCommand {
    private int sno;
    private String sname;
    private String sadd;
    public int getSno() {
            return sno;

    }
    public void setSno(int sno) {
            this.sno = sno;
    }
    public String getSname() {
            return sname;
    }
    public void setSname(String sname) {
            this.sname = sname;
    }
    public String getSadd() {
            return sadd;
    }
    public void setSadd(String sadd) {
            this.sadd = sadd;
    }
}
```

PerformOperationsController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.multiaction.MultiActionController;
import com.nt.command.TestCommand;
public class PerformOperationsController extends MultiActionController {

    public ModelAndView insert(HttpServletRequest req,HttpServletResponse
res,TestCommand    cmd){
        System.out.println(cmd.getSno());
            return new ModelAndView("user","operation","Insert Operation");
    }
            public ModelAndView update(HttpServletRequest req,HttpServletResponse
    res,TestCommand cmd){
            return new ModelAndView("user","operation","Update Operation");
    }
    public ModelAndView delete(HttpServletRequest req,HttpServletResponse
res,TestCommand    cmd){
    return new ModelAndView("user","operation","delete Operation");
```

```
    }

  public ModelAndView invalid(HttpServletRequest req,HttpServletResponse res,TestCommand
      cmd){
      return new ModelAndView("user","operation","InValid Operation");
      }
      }
```

index.jsp:

```
<jsp:forward page="form.htm"/>
```

web.xml:

```
<web-app>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/form.htm">pvc1</prop>
<prop key="/perform.htm">poc</prop>
</props>
</property>
</bean>
<bean id="pvc1"
    class="org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="user"/>
```

```
</bean>
<bean id="mnr"
    class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodName
Resolver">
<property name="paramName"  value="opt"/>
<property name="defaultMethodName"  value="invalid"/>
</bean>
<bean id="poc"  class="com.nt.controller.PerformOperationsController">
<property name="methodNameResolver"  ref="mnr"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

user.jsp:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<form  action="perform.htm" method="post">
  Sno : <input type="text"  name="sno"/><br>
  Sname :<input type="text"  name="sname"/><br>
  Sadd :<input type="text"  name="sadd"/><br>
<input type="submit" name="opt" value="insert"/>
<input type="submit" name="opt" value="update"/>
<input type="submit" name="opt" value="delete"/>
<input type="submit" name="opt1" value="view"/>
</form>
<c:if test="${operation ne null }">
<c:out value="${operation}"/>  Successspringframeworkul
</c:if>
```

Note:

- Only SimpleFormController, AbstractCommandController classes are removed from spring 4.0

While working with MultiActionController there is no need of configuring command class any where because the superclass handlerRequest (-,-) internally uses Reflection API to take specified 3rd parameter type or 4th parameter type (if 3rd parameter is HttpSession) as command class name...

AbstractWizardFormController:

- So far we worked with single form page based controller classes like SimpleFormController, MultiActionController and etc... In order to work with multiple form pages that comes in a

sequence having wizard style we need to use **AbstractWizardFormController** by specifying list form pages to display in a sequence.In this controller the data of all the form pages will come and store in single command class object. Based on the name of additional request parameter that comes comes along with the request this will display the next form page or process the request.

- ⇨ _cancel: leaves the wizard without performing final action (cancel button)
- ⇨ _finish: leaves the wizard by performing final Action (finish button)
- ⇨ _target<n>: displays the nth page from the list of form pages that are configured (for previous,next buttons)

Flow:

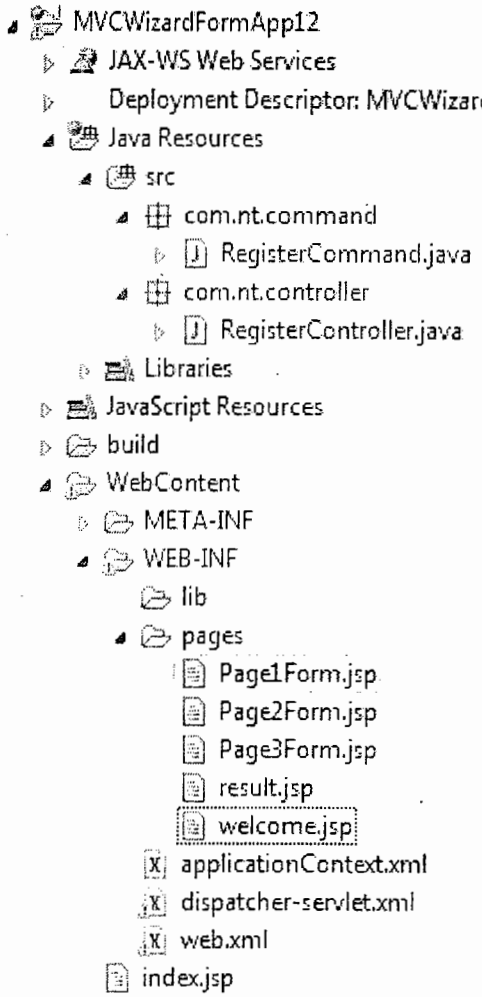
Welcomepage-->formpage1---->formpage2------>formpage3----->Result page.
We can override the following methods in our AbstractWizardFormController.

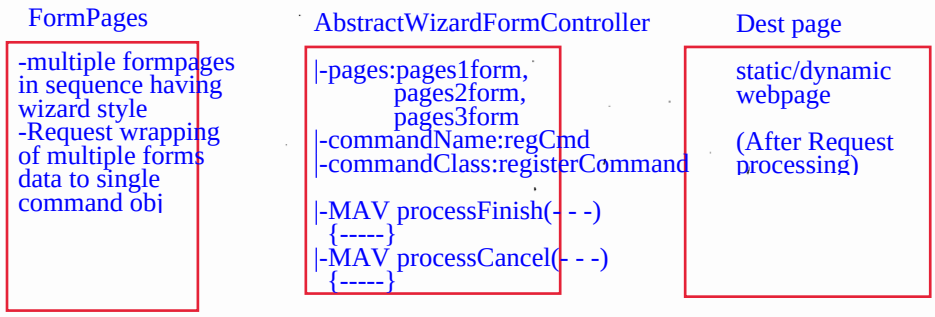
formBackingObject(-): creates and returns commands object
proccessFinish(..): takes final action of request processing for finish button.
processCancel(-): executes the logic related to "Cancel button".

```
index.jsp
<jsp:forward page="home.htm">

welcome.jsp
<a href="register.htm">
  register
</a>

page1Form.jsp
------------------
<form:form method="POST">
  ...
  ...
  <input type="submit"
         name="_targe1"
         value="next">
</form>

page2Form.jsp
------------------
<form method="POST">
  ....
<input type="submit" name="_target2" value="next">
<input type="submit" name="_target0" value="previous">
<input type="submit" name="_cancel" value="cancel">
</form>

page3Form.jsp
<form:form method="POST">
  ...
  ..
<input type="submit" name="_finish" value="finish">
<input type="submit" name="_target1" value="previous">
<input type="submit" name="_cancel" value="cancel">
</form:form>
```

```
DispatcherServlet
  *.htm

View obj

InternalResourceViewResolver
  |--->prefix: /WEB-INF/pages/
  |--->suffix : .jsp

result.jsp
  ....
  .....
  .....
```

```
SimpleUrlHandlerMapping
  |-->[home.htm<-->pvc1]
  |-->[register.htm<--->rc]

pvc1
ParameterizableViewController
  |---> viewName:welcome

RegisterController (AWFC)
  |--->pages
      |-->page1Form.
      |-->page2Form.
      |-->page3Form.
  |--->commandName: regCmd

formBackingObject(-){
  return new RegisterCommand();
}

processFinish(req,req,cmd,BE){
  ...
  ... new
return MAV(-,-,-);
}

processCancel(req,res,cmd,BE){
  ....
  ...
return new MAV(-,-,-);
}
```





_target<n>----:displays nth page(give_target<n> for previous,next buttons)
_finish-->processFish(-,-,-)(give_finish as name of the finish button)
_cancel-->processCancel(-,-,-,)(give_cancel as the name of cancel)

Add Jars: add 3.x jar files

RegisterCommand.java:

```
package com.nt.command;
public class RegisterCommand{
        //Login Credentials
    private String userName;
    private String password;
    //Contact Details
    private String emailId;
    private String phone;
    //Address Details
    private String city;
    private String pincode;
    public RegisterCommand(){
        System.out.println("RegisterCommand:0-param constructor");
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
```

```
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public String getEmailId() {
		return emailId;
	}
	public void setEmailId(String emailId) {
		this.emailId = emailId;
	}
	public String getPhone() {
		return phone;
	}
	public void setPhone(String phone) {
		this.phone = phone;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	public String getPincode() {
		return pincode;
	}
	public void setPincode(String pincode) {
		this.pincode = pincode;
	}
	@Override
	public String toString() {
		return "RegisterCommand [userName=" + userName + ", password="
				+ password + ", emailId=" + emailId + ", phone=" + phone
				+ ", city=" + city + ", pincode=" + pincode + "]";
	}
}
```

RegisterController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractWizardFormController;
import com.nt.command.RegisterCommand;
```

```
public class RegisterController extends AbstractWizardFormController{
    /*public RegisterController(){
            setCommandName("userForm");
    }*/
    }
    @Override
    protected ModelAndView processfinish(HttpServletRequest request,
                    HttpServletResponse response, Object command, BindException errors)
                    throws Exception {
    System.out.println("RegisterController.processpringframeworkinish(-,-,-,-)");
            //Get the data from command object
            RegisterCommand cmd = (RegisterCommand)command;

            //where is the finish page?
            return new ModelAndView("result", "res", cmd);
    }
    @Override
    protected ModelAndView processCancel(HttpServletRequest request,
                    HttpServletResponse response, Object command, BindException errors)
                    throws Exception {
            System.out.println("RegisterController.processCancel(-,-,-,-)");
            //where is the cancel page?
            return new ModelAndView("welcome");
    }
}
```

index.jsp:

```
<jsp:forward page="home.htm"/>
```

web.xml:

```
<web-app>
<servlet>
    <servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Dispatcher-servlet.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-2.5.xsd">

<bean id="surl"  class="org.springframework.web.servlet.handler.SimpleUrlHandler
Mapping">
<property name="mappings">
<props>
<prop key="/home.htm">pvc1</prop>
<prop key="/register.htm">rc</prop>
</props>
</property>
</bean>
<bean id="pvc1"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="welcome"/>
</bean>

<bean id="rc" class="com.nt.controller.RegisterController" >
<property name="commandName" value="regCmd"/>
    <property name="pages">
            <list>
                    <value>Page1Form</value>
                    <value>Page2Form</value>
                    <value>Page3Form</value>
                    </list>
            </property>
</bean>
    <bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
<property name="prefix" value="/WEB-INF/pages/"/>
<property name="suffix" value=".jsp"/>
</bean>
</beans>
```

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

Page1Form.jsp:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<form:form method="POST" commandName="regCmd">
Username : <form:input path="userName"/><br>
Password:  <form:password path="password"/><br>
<input type="submit" value="Next" name="_target1" />
<input type="submit" value="Cancel" name="_cancel" />
</form:form>
```

Page2Form.sjp:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<form:form method="POST" commandName="regCmd">
Email :<form:input path="emailId"/><br>
Phone:<form:input path="phone" /><br>
<input type="submit" value="Previous" name="_target0" />
<input type="submit" value="Next" name="_target2" />
<input type="submit" value="Cancel" name="_cancel" />
</form:form>
```

Page3Form.jsp:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<form:form method="POST" commandName="regCmd">
City : <form:input path="city" /><br>
Pincode:<form:input path="pincode" /><br>
<input type="submit" value="Previous" name="_target1" />
<input type="submit" value="Finish" name="_finish" />
<input type="submit" value="Cancel" name="_cancel" />
</form:form>
```

result.jsp:

```
<h1><center> Result page</center></h1>
${res}
```

welcome.jsp:

```
<H1><center> HOME PAGE</center></H1>
<br>
<br>
<a href="register.htm"> Register</a>
```

Note:-

- When we give initial phase request to AbstractWizardFormController it internally calls FormBackingObject() throughhandleRequest(-,-) method to create command object and to keep that object in session scope.

- When postback request is given having _target<n> request parameter name it displays next or previous page postback request having the request param name _cancel calls processCancel(-) through handleRequest(-) method.

- Similarly calls processFinish(-) for postback request that contains _finish request param.

=========0000000=========

HandlerMapping:(Different types of HandlerMappings)

- Handler Mapping is a component that maps incoming request to controller/handler class. All handlermapping classes are the implementation classes of "HandlerMapping" (i).

The important HandlerMapping classes are:

1) BeanNameUrlHanlderMapping (default for xml configuration)
2) SimpleUrlHandlerMapping
3) ControllerClassNameHandlerMapping
4) DefaultAnnotationHandlerMapping (default for annotation configration)

1) BeanNameUrlHandlerMapping:

- This maps incoming request to controller class based on the servlet path or virtual path of incoming request url. i.e the virtual path of incoming request url must be taken as bean name of controller class.

eg:

```
<bean class="org.springframework.web.servlet.hanlder.BeanNameUrlHandlerMapping"/>
<bean name="/home.htm" class="com.nt.controller.WelcomeController"/>
```

- If the incoming request url contains "/home.htm" then it is maps the request to WelcomeController class.

2) SimpleUrlHandlerMapping:

- This is useful to map the incoming requests with controller classes based on virtual paths of request urls and bean ids/names of controller class.

We need to supply the details as values of "mappings" property of type (java.util.Properties)

eg:

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<props>
<prop key="/home.htm">wc</prop>
</props>
</property>
</bean>
<bean name="wc" class="com.nt.controller.WelcomeController"/>
```

If request url contains "/home.htm" then it passes the request to WelcomeController class whose bean id /name as"wc".

3) ControllerClassNameHandlerMapping:

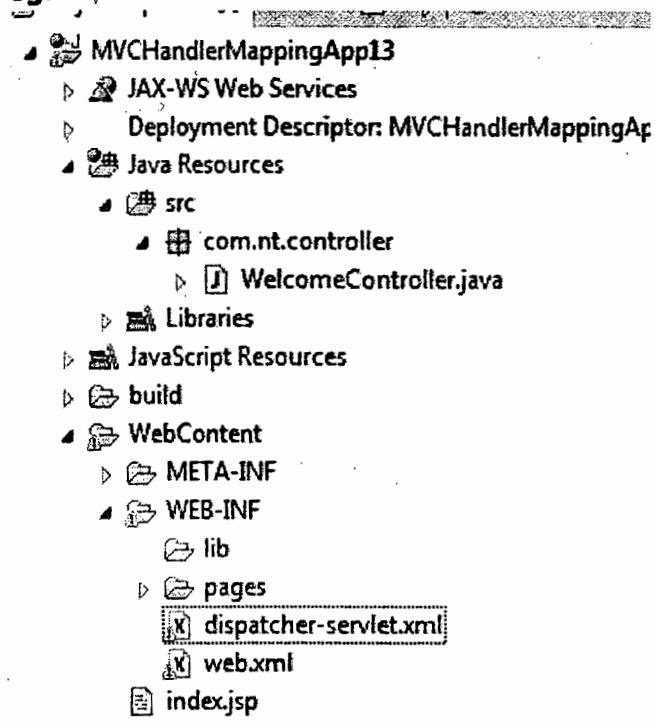
- Maps incoming request with controller classes based on virtual paths and controller class names. note:-using this hander mapping we can not map incoming request url with predefine controller classes like parameterizable view controller,urlfilename view controller

- It takes virtual path, removes extension, makes firstLetter as uppercase letter and appends Controller word to it and looks for controller class whose name is that name.

Note:

If request url contains "welcome.htm" then it maps the request to a controller class whose name is "WelcomeController".

eg:



WelcomeController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
public class WelcomeController implements Controller {
    @Override
    public ModelAndView handleRequest(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
            return new ModelAndView("welcome","msg","Welcome To Spring MVC");
    }}
```

index.jsp

```
<jsp:forward page="welcome.htm"/>
```

web.xml:

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean  id="wc" class="com.nt.controller.WelcomeController"/>
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="welcome.htm">wc</prop>
</props>
</property>
<property name="order" value="1"/>
</bean>
<bean
    class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandler
Mapping">
<property name="order"  value="2"/>
</bean>
<bean id="vr"  class="org.springframework.web.servlet.view.InternalResourceView
Resolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

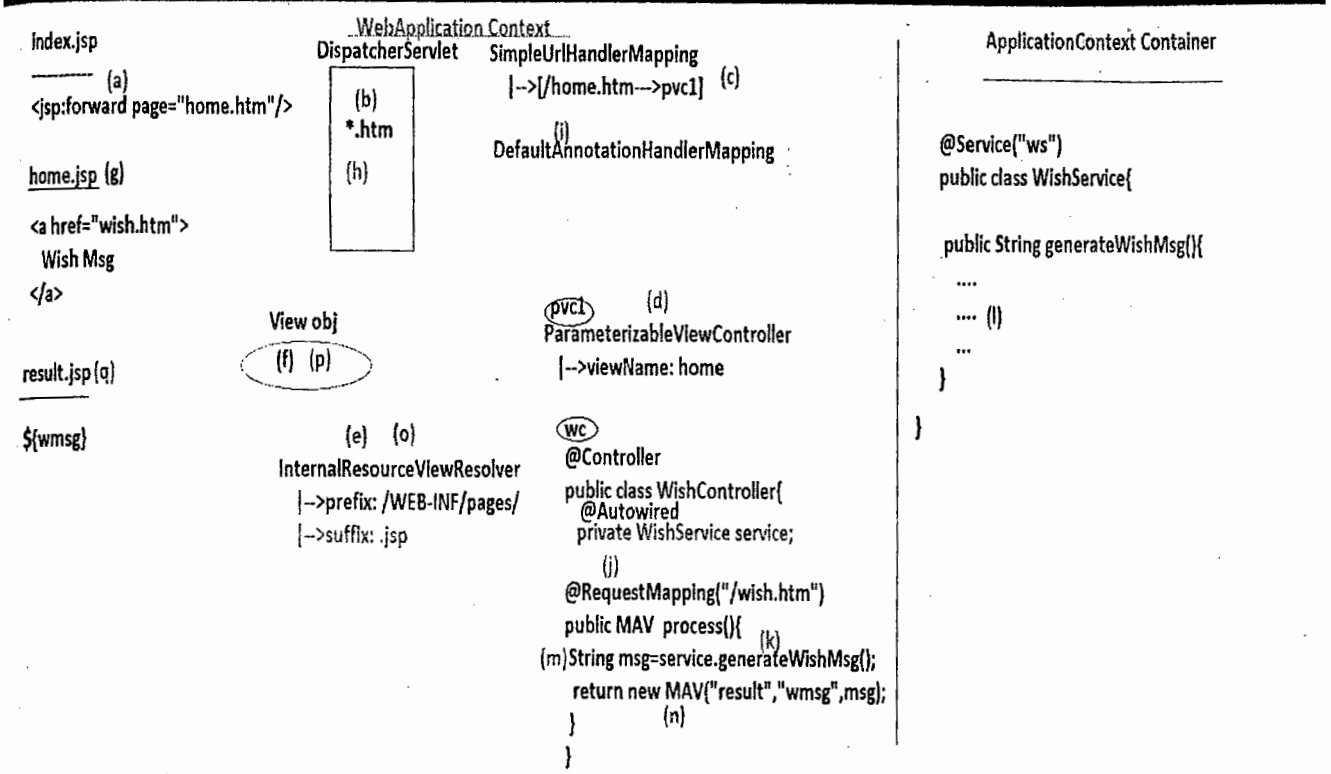
welcome.jsp:

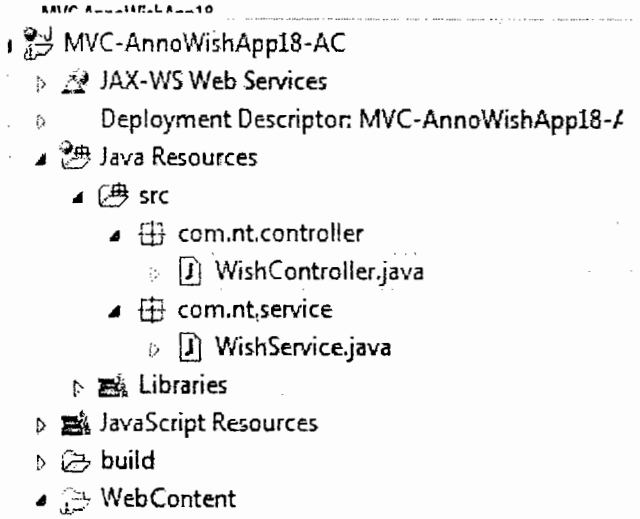
```
<h1><center> HOME PAGE</center></h1>
${requestScope.msg}
```

DefaultAnnotationHandlerMapping:

Maps the incoming request url with controller class methods based on @RequestMapping annotation info.

Diagram:





WishService.java

```
package com.nt.service;
import java.util.Calendar;
import javax.inject.Named;
@Named("ws")
public class WishService {
    public String generateWishMsg(){
        // generate wish Msg based on current hour the day
        Calendar cl=Calendar.getInstance();
```

```
            int hour=cl.get(Calendar.HOUR_OF_DAY);
            if(hour<=12)
                    return  "Good morning";
            else if(hour<=16)
                    return "Good After noon";
            else if (hour<=20)
              return "Good Evening";
            else
                    return "Good Night";
    }//method
}//class
```

WishController.java

```
package com.nt.controller;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import com.nt.service.WishService;
@Controller
public class WishController {
    @Resource
    private WishService service;
/*  @RequestMapping("/wish.htm")
    public  ModelAndView process(){
            //use service class
            String msg=service.generateWishMsg();

            return new ModelAndView("result","wmsg",msg);
    }//process */
    /*      @RequestMapping("/wish.htm")
    public  String  process(Model model){

            //use service class
            String msg=service.generateWishMsg();

            // add data to mode
            model.addAttribute("wmsg",msg);
            return "result";
    } */
    /*      @RequestMapping("/wish.htm")
    public  String  process(ModelMap map){
            //use service class
            String msg=service.generateWishMsg();
                    map.addAttribute("wmsg", msg);
            return "result";
    } */
/*  @RequestMapping("/wish.htm")
```

```
    public String process(Map<String,Object> map){
            //use service class
            String msg=service.generateWishMsg();
            map.put("wmsg",msg);
            return "result";
    } */
    @RequestMapping("/wish.htm")
    public void process(Model model){
            //Use service class
            String msg=service.generateWishMsg();
            model.addAttribute("wmsg",msg);
    }
}//class
```

index.jsp

```
<jsp:forward page="home.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
```

```
    <props>
    <prop key="/home.htm">pvc1</prop>
    </props>
    </property>
    </bean>
    <bean id="pvc1"
        class="org.springframework.web.servlet.mvc.ParameterizableViewController">
    <property name="viewName"  value="home"/>
    </bean>
    <bean      class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotation
    HandlerMapping"/>
    <!-- <bean        class="org.springframework.web.servlet.view.DefaultRequest
    ToViewNameTranslator"/> -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix"  value="/WEB-INF/pages/"/>
    <property name="suffix"  value=".jsp"/>
    </bean>
    <context:component-scan base-package="com.nt.controller"/>
</beans>
```

applicationContext.xml

```
    <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
    <context:component-scan base-package="com.nt.service"/>
    </beans>
```

Home.jsp

```
    <h1><center> Welcome to Annotations Based Spring MVC</center></h1>
    <a href="wish.htm"> wish Msg</a>
```

Result.jsp

```
    <center><h1> Result Page</h1></center>
    <br>
    Wish Message :  ${wmsg}
    <a href="home.htm">home</a>
```

Wish.jsp

```
    <b> wish.jsp</b>
    Result is : ${wmsg}
```

⇨ When we configure multiple handler mapping in which order they should be executed will be decided based on "order" priority value high value indicates low priority and low value indicates high priority. if order priority is not specified priority will be given to the first configure handler mapping

Request: welcome.htm

⇨ eg:

```
<bean class="org.springfraemwork.web.saervlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
</property>
<property name="order" value="1"/>
</bean>
<bean
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping">
<property name="order" value="2"/>
</bean>
```

While configuring handler mapping we can also specify the order and handler intercepter.

HandlerInterceptor:

⇨ It act as hook (additional) for controller/handler class to have pre and post processing logics. It is like SrvletFilter of J2EE web application development, But it is different from filter in 1 way. Filter can have only pre request processing and post response processing logics. Where as handler interceptor can have pre-request processing, before rendering view, after rendering view logics with respect to controller classes.

⇨ To develop interceptor we need to take a class that implements "HandlerInterceptor" (i), But there is adaptor class for this whose name is "HandlerInterceptorAdapter", so we can extend from this class to override our choice methods.

preHandle(--)------>executes before control goes to controllerclass

postHandle(--)------>executes once controller class finished the execution but before rendering the view.

afterCompletion() : executes after rendering the view.

Usecases of HandlerInterceptor:

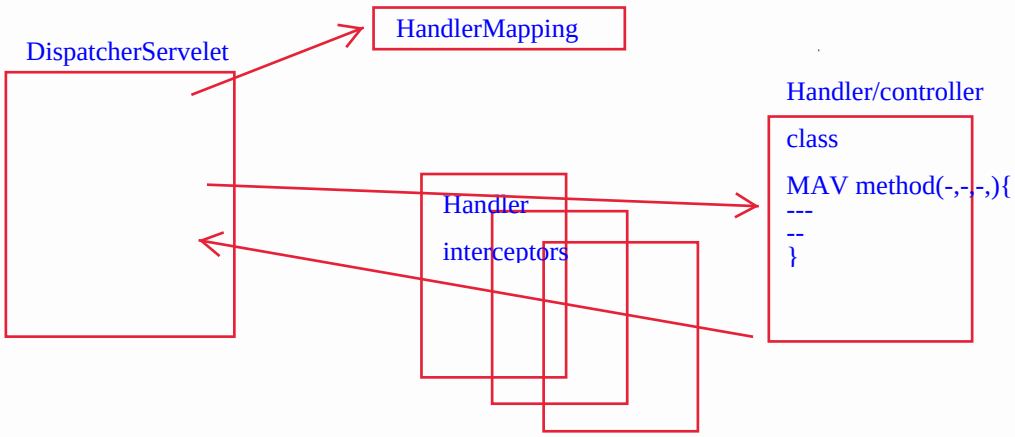
We can develop HandlerInterceptor class for controller only to take the request between 9am to 5pm.

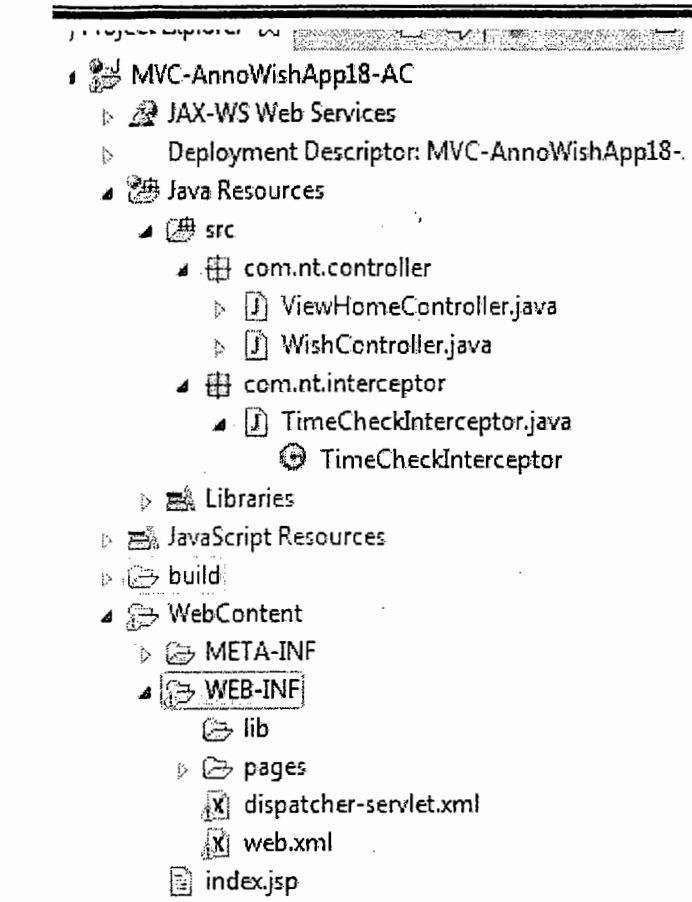
We can develop Handlerinterceptor class for controller only to allow the request that are coming from certain browser and etc.... take the support of hander interceptor of to apply optional services as pre or post request processing service on controller,hander classes.this are work like servlet filter.

Steps:-

1) Keep any spring application ready.
2) Develop HandlerInterceptor class.
3) Specifying HandlerInterceptor has the dependent value to HandlerMapping.

```
<bean class="pkg.BeanNameUrlHadlerMapping"/>
```




MVC-AnnoWishApp18-AC
JAX-WS Web Services
Deployment Descriptor: MVC-AnnoWishApp18-.
Java Resources
src
com.nt.controller
ViewHomeController.java
WishController.java
com.nt.interceptor
TimeCheckInterceptor.java
TimeCheckInterceptor
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
pages
dispatcher-servlet.xml
web.xml
index.jsp

TimeCheckInterceptor.java:

```
package com.nt.interceptor;
import java.util.Calendar;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
public class TimeCheckInterceptor extends HandlerInterceptorAdapter {
    @Override
    public boolean preHandle(HttpServletRequest request,
                    HttpServletResponse response, Object handler) throws Exception {
            Calendar cl=Calendar.getInstance(); //sys date and time
            int hour=cl.get(Calendar.HOUR_OF_DAY); //current hour in 24hr format
            if(hour<9 || hour>=17){
                    response.sendRedirect("timeout.jsp");
                    return false;
            }
            return true;
    }//preHandle
}//class
```

ViewHomeController.java:

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class ViewHomeController extends AbstractController {
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {

            return new ModelAndView("home");
    }
}
```

WishController.java

```
package com.nt.controller;
import java.util.Calendar;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class WishController extends AbstractController {
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
            // get System date and time
            Calendar cl=Calendar.getInstance();
            //get current hour of the day
```

```
            int hour=ci.get(Calendar.HOUR_OF_DAY);
            String msg=null;
            if(hour<12)
                    msg="Good morning";
            else if(hour<16)
                    msg="Good afternoon";
            else if(hour<20)
                    msg="Good evening";
            else
                    msg="Good Night";
        return new ModelAndView("result","wmsg",msg);
        }
    }
```

Index.jsp:

```
<jsp:forward page="home.htm"/>
```

Tmeout.jsp

```
<span style="color:red">Request must be given in between 9am to 5 pm</span>
```

AboutUs.jsp

```
NIT, the poineer in IT Training, got started 13 years back in the year
2003.
```

ContactUs.jsp

```
NIT,
opp Satyam threater
Ammerpet
```

Home.jsp

```
<b><center>  Home page</center></b>     <a
    href="AboutUs.htm">AboutUs</a>     <a
    href="ContactUs.htm">ContactUs</a>
<br>
<a href="wish.htm">Wish Message</a>
```

Result.jsp:

```
<b><center> Result Page</center></b><br>
Wish Message : ${wmsg}.
<br>
<a href="home.htm">home</a>
```

dispatcher-servlet.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-3.0.xsd">
<bean  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="home.htm">vhc</prop>
<prop key="AboutUs.htm">ufv</prop>
<prop key="ContactUs.htm">ufv</prop>
</props>
</property>
</bean>
<bean  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="wish.htm">wc</prop>
</props>
</property>
<property name="interceptors">
<list>
<ref bean="timeCheckInterceptor"/>
</list>
</property>
</bean>
<bean id="timeCheckInterceptor"  class="com.nt.interceptor.TimeCheckInterceptor"/>
<bean id="vhc"  class="com.nt.controller.ViewHomeController"/>
<bean id="wc"  class="com.nt.controller.WishController"/>
<bean id="ufv" class="org.springframework.web.servlet.mvc.UrlFilenameViewController"/>
<!--   <bean id="pvc1"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="AboutUs"/>
</bean>
<bean id="pvc2"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="ContactUs"/>
</bean> -->
<bean id="vr"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
    </web-app>
```

ViewResolver:

- This component takes logical view name given by controller through DispatcherServlet and returns "ViewObject" (The implementation class object of View (i)) pointing to physical view.DispatcherServlet calls render (-,-) method on this view object to render the control to view resource.
- Like html file, jspfile, bean class . all ViewResolvers implements "ViewResolver" interface. Spring has supplied few predefined viewResolver classes. They are

a) InternalResourceViewResolver:

- Useful to resolve jsp, servlet programs of WEB-INF folder as views
  By default takes "JstlView" as view class

b) UrlBasedViewReslover:

- Super class for (a) and we must specify view class explicitly

c) ResorceBundleViewResolver:

- Allows to configure view in properties file.

d) XmlViewResolver:

- Allows to configure views in xml file.

e) BeanNameViewResolver:

- Useful to resolver spring beans as views.

UrlBasedViewResolver:

- Takes logical viewname from controller class and searches for view resources whose name is same as logical viewname. We must configure "viewClass" explicitly while configuring this view Resolver class.
- This viewResolver can be used to resolve any view like html, jsp beans class and etc... by configuring an appropriate View class (Viewclass configuration is mandatory)
- We configure JstlView as view class to make ViewResolver to resolve Html or Servlets or jsp

prgms as views i.e ViewResolver gives JstlView class object and render() will be called on that object to render the view to browser.

does not take default view class but allow to configure any view class to render view object having any technology viewresource the configureble view classes are JstlView InternalResourceView,TillesView etc

```
<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
<property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
<property name="suffix" value=".jsp"/>
<property name="prefix" value="/WEB-INF/Pages/">
</bean>
```

use internalResourceview when jsp are our view without using jstl tag and use Jstl View when jsp and also we need to add jstl.jar,standerd.jar files

what is the difference between internalResourceViewResolver and UrlbaesdViewResolver?
:--UrlBased ViewResolver allow us to take any view class to rendor the view in any technology.that means view class configuration is mandatory here.

InternalResourceViewResolver is subclass of Urlbased View Resolver that take internal and allow us to rendor servlet jsp components that are placed inside web-inf folder.here View class are optional

InternalResourceViewResolver: InternalResourceView

- It is the subclass of UrlBasedViewResolver, that takes ~~JstlView~~ as default view class having capability to resolve the serlvet, jsp progs of web application that are there under WEB-INF folder.
- While working with with ViewResolver to render jsp file as ViewResource without jstl tags then use InternalResourceView class. To render jsp with jstl tags then use ~~JstlView~~ class
- It is most regularly used viewresolver. InternalResourceView

```
<bean class="org.springframework.web.servlet.InternalRourceViewResolve">
<property name="prefix" value="/WEN-INF/Pages"/>
<property name="suffix" value=".jsp/">
</bean>
```

ResourceBundleViewResolver:

- The problem with above 2 view Resolver is logical view name and jsp file name must match. If jsp filename is changed later then modifying logical viewname by opening the source code of controller class is very difficult.
- In order to overcome this problem we canuseResourceBundleViewResolver where we can configure views in the propertiesfile.

Views.properties:

- Default name is also views.properties

(src folder)

```
welcome.(class)=org.springframework.web.servlet.view.Jstlview
(logical view name)
welcome.url=/WEB-INF/pages/namaskar.jsp
(physical view name)
```

In dispatcherservlet: if no viewresolver is configured the default view resolver is internalResourceViewResolver

- <bean id="**vr**"
class="org.springframework.web.servlet.view.ResourceBundleViewResolver">

```
<property name="basename" value="views"/>
</bean>
```

- We can use ResourceBundleViewResolver to enable internationalization on our spring MVC applications.
- We can take multiple jsps for multiple locales and we can configure them in multiple locale specific properties file.

XmlViewResolver:

- Similar to ResourceBundleViewResolver, but allows to take xml file to specify view configrations. Takes "view.xml" as the default name. we need to specify this file name as the value of "location" property of XmlViewResolver in configuration file.
- eg:

views.xml (WEB-INF folder)

```
<bean id="welcome" class="org.springframework.web.servlet.view.JstlView">
<property name="url" value="/WEB-INF/pages/salam.jsp"/>
   In dispatcher-servlet.xml
<bean class="org.springframewok.web.Servlet.view.XmlViewResolver"/>
<property name="location" value="/WEB-INF/pages/views.jsp"/>
</bean>
```

BeanNameViewResolver:

- Resolves java class/spring bean as the viewResource, for this the logical view name gathered from controller class must be taken as bean id or name for the class/bean that is acting as view.

dispatcher-servlet.xml

- <!----Java class/Spring class acting as view----->

```
<bean name="welcome" class="com.nt.view.MyReportExcel"/>
<bean id="ur" class="org.springframework.web.servlet.view.BeanNameviewResolver"/>
```

~~Working with Views:~~

- When we configure multiple viewResolvers they will execute in a chain.

InternalResourceViewResolver always gets high priority when compare to other ViewResolvers in chain so it is always recommended to configure InternalResourceViewResolver as the last ViewResolver in the chain.

Working with Views:--

- Spring allows to take different technologies in the view layerlike jsp,html,servlet,velacity, freemarker and etc... In spring we can take java classes/spring beans as view by extending them from Abstractxxxview classes (like AbstractPdfView, AbstractExcelView.....) To render response as pdf,excel and etc docs.

In spring view is an abstract entity so we can take anything as view.
ViewResolver gives "View object" (View(i) implementation class object) and DispatcherServlet calls render(-) on the object. This render(-) method passes the control to physical view Files/calss (like the jsp or above discussed classes).

To develop view we should take class implementing View(i). but some predefined Abstract and Concreate view classes are supplied to simplify the process. They are JstlView, AbstractPdfView, AbstractExcelView etc...

Example app on BeanNameViewResolver and java classes as views:

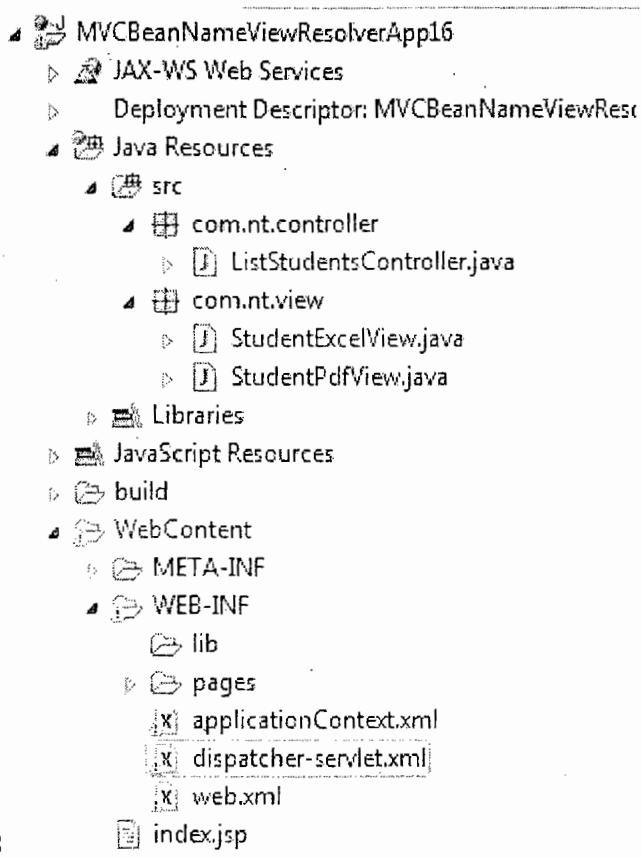
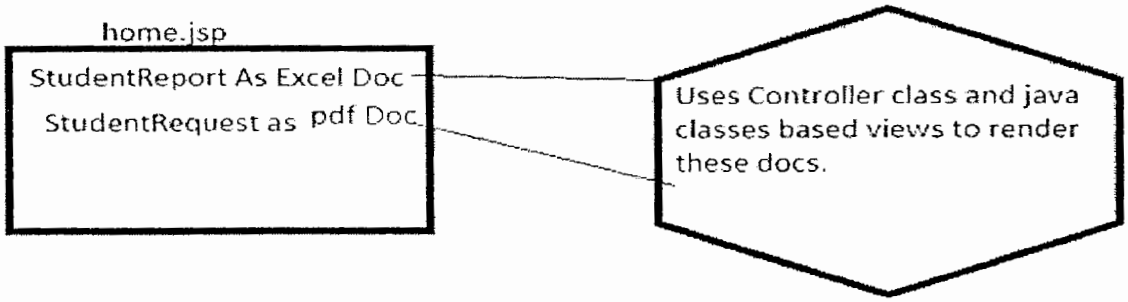


Diagram:



NOTE:

We can use poi api Support to render responseas **Excel**doc. We can use itext api support to render responce as **Pdf doc**
=> Poi in an 3rd party api given by apache to render the responce as Excelsheet...

Jars:-

```
MVC library
Standard.jar
Jstl.jar
iText-2.0.7.jar(-->Spring 2.5)---> higher version for spring 4  }
Poi-3.0.1.jar                                                    }Gather them from
spring 2.5
```

ListStudentsController.java

```
package com.nt.controller;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class ListStudentsController extends AbstractController {
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                HttpServletResponse res) throws Exception {
        List <String> list=new ArrayList<String>();
        list.add("raja"); list.add("ravi"); list.add("jani");
        if(req.getParameter("report").equals("pdf"))
                return new ModelAndView("pdfView","stList",list);
        else
                return new ModelAndView("excelView","stList",list);
    }
}
```

StudentExcelView.java

```
package com.nt.view;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import
org.apache.poi.hssspringframework.usermodel.HSSPRINGFRAMEWORKRichTextString;
import org.apache.poi.hssspringframework.usermodel.HSSPRINGFRAMEWORKSheet;
import org.apache.poi.hssspringframework.usermodel.HSSPRINGFRAMEWORKWorkbook;
import org.springframework.web.servlet.view.document.AbstractExcelView;
public class StudentExcelView extends AbstractExcelView{
    @Override
    protected void buildExcelDocument(Map<String, Object> map,
                HSSFWorkbook book, HttpServletRequest req,HttpServletResponse res)
                throws Exception {
        List list =(ArrayList)map.get("stList");
        HSSFSheet sheet=book.createSheet("Sheet1");
        getCell(sheet,0,0).setCellValue(new HSSFRichTextString("Students     Names"));
        getCell(sheet,1,0).setCellValue(new        HSSFRichTextString(list.get(0)+" "));
        getCell(sheet,2,0).setCellValue(new        HSSFRichTextString(list.get(1)+" "));
        getCell(sheet,3,0).setCellValue(new        HSSFRichTextString(list.get(2)+" "));
    }
}
```

StudentPdfView.java

```
package com.nt.view;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.view.document.AbstractPdfView;
```

```
import com.lowagie.text.Document;
import com.lowagie.text.Paragraph;
import com.lowagie.text.Table;
import com.lowagie.text.pdf.PdfWriter;
public class StudentPdfView extends AbstractPdfView {
    @Override
    protected void buildPdfDocument(Map<String, Object> map, Document doc,
                PdfWriter writer, HttpServletRequest req, HttpServletResponse res)
                throws Exception {
        //read Model object attribute value
        List list =(ArrayList)map.get("stList");
        Paragraph p = new Paragraph("Student details");
        p.setAlignment("center");
        doc.add(p);
        Table t = new Table(1);
        t.addCell(list.get(0)+" ");
        t.addCell(list.get(1)+" ");
        t.addCell(list.get(2)+" ");
        doc.add(t);
    }
}
```

index.jsp

```
<jsp:forward page="home.htm"/>
```

web.xml

```
<web-app>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.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-3.0.xsd">
<bean id="surl"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="home.htm">pvc1</prop>
<prop key="listStudents.htm">lsc</prop>
</props>
</property>
</bean>
<bean id="lsc" class="com.nt.controller.ListStudentsController"/>
<bean id="pvc1"  class="
org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="home"/>
</bean>
<bean id="excelView"  class="com.nt.view.StudentExcelView"/>
<bean id="pdfView"  class="com.nt.view.StudentPdfView"/>
<bean id="bnvr"  class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
<bean id="vr"  class="org.springframework.web.servlet.view.InternalResource
ViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
</beans>
```

applicationContext.xml

```
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:util="http://www.springframework.org/schema/util"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

Home.jsp

```
<b><center>  Home page</center></b>
<br>
<a href="listStudents.htm?report=pdf">Student Report as Pdf</a>
<br>
<br>
<a href="listStudents.htm?report=excel">Student Report as Excel</a>
```

i18n (Internationalization)

- Making our application working for different locales by rendering lebels in different languages is called enabling i18n on the application.This improves no of customer/visitors

to the application. For this we need to take multiple properties files for multiple locales on 1 per local basis like welcome.properties, welcome_fr_FR.properties, welcome_zh_cn.properties and etc...

- To enable i18n on spring mvc application we need to configure LocaleResolver, LocaleChangeInterceptor classes as beans and we need to pass interceptor reference to link interceptor with controller class.
- SessionLocaleResolver is implementation class of LocaleResolver interface that allows to configure locale explicitly otherwise fallbacks to default locale configured or accept-language header value. This also enables **i18n** on the application

```
<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
<property name="defaultLocale" value="en"/>
</bean>
```

LocaleChangeInterceptor allows to change locale for every request with respect to controller based on the additional request param that is configured.

```
<bean id="lci" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
<property name="paramName" value="language"/>
</bean>
```

- Based on the value of language request parameter the locale will be changed for every request by overrdeing default locale.
- Based on this locale properties file will be picked up to render the label. While preparing multiple properties files in the application of internationalization, all propertiespringframeworkiles must have same keys and different lebels specific to each locale.

Steps to develop I18N based spring app:

1) Prepare multiple properties files for multiple locales including one base properties file.

Note: make sure that allproperties files the keys are same.

2) Configure properties file in dispatcher-servlet.xml file using "ResourceBundleMessageSource" having fixed beanid (**messageSource**).

3) Configure SessionLocaleResolver specifying the default locale name.

4) Configure LocaleChangeInterceptor as spring bean and link the above interceptor with controller class that renders the view.

5) Add **spring:message** tag in jsp to get the lables from the appropriate properties file.

eg:

- MVCI18NApp17
  - JAX-WS Web Services
  - Deployment Descriptor: MVCI18NApp17
  - Java Resources
    - src
      - com.nt.controller
      - com.nt.view
      - welcome_zh_CN.properties
      - welcome.properties
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
      - pages
      - dispatcher-servlet.xml
      - web.xml
    - index.jsp

welcome.properties

my.un= Enter UserName

my.pwd=Enter Password

welcome_zh_CN.properties

my.un=\u0917\u094a\u092a\u093f\u0905\u0928\u094d

my.pwd=PICHACHU

WelcomeController.java (q)

```
package com.nat.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class WelcomeController extends AbstractController{
        @Override                    (h)  (r)
    protected ModelAndView handleRequestInternal(HttpServletRequest request,
                HttpServletResponse response) throws Exception {

        ModelAndView model = new ModelAndView("Register");

        return model;  (i)  (s)
    }
}
```

redirect.jsp (a)

```
<jsp:forward page="welcome.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
(c) <servlet-name>dispatcher</servlet-name>                    (m)
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>redirect.jsp</welcome-file>   (b)
</welcome-file-list>
</web-app>
```

dispatcher-servlet.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-2.5.xsd">
    <bean id="localeResolver"
    class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
            <property name="defaultLocale" value="en" />
    </bean>     (N)     f// since language request param is not there in the request it takes "EN" as the default locale as per
(p) <bean id="localeChangeInterceptor"   configuration done in session locale resolver.
based on    class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
lang param          <property name="paramName" value="language" />
it takes    </bean>
ZH-CN               <!-- if url=/welcome,it will call WelcomeController -->
as locale   <bean          class="org.springframework.web.servlet.mvc.support.Controller
value
ClassNameHandlerMapping" >
            <property name="interceptors">
                    <list>
                            <ref bean="localeChangeInterceptor" />
                    </list>
            </property>
</bean>
    <!-- Register the bean -->
    <bean class="com.nat.controller.WelcomeController" />   (g)
<!-- Register the welcome.properties -->
    <bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
            <property name="basename" value="welcome" />
    </bean>
```

```
    <bean id="viewResolver"(j)(t)
    class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
  <property name="prefix">
  <value>/WEB-INF/pages/</value>
  </property>
  <property name="suffix">
  <value>.jsp</value>
  </property>
  </bean></beans>
```

(U) <spring:msg> reads the table values from welcome-zh-CN.properties

register.jsp (k) //here <spring message > actually looks for welcom-EN property since not available collects the lables from base properties file welcom.properties

```
<%@taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@page isELIgnored="false" %>
Registration Page<br>
 Language : <a href="?language=en">English</a>|
<a href="?language=zh_CN">Chinese</a>
<h3>
    <spring:message code="my.un" text="default text" />
    <input type=text name="userName"><br>
    <spring:message code="my.pwd" text="default text" />
    <input type=text name="password"><br>
</h3>
 Selected Locale: ${pageContext.response.locale}
</body>
</html>
```

(l)

Annotation Driven Spring MVC:

- In xml configuration based programmatic approach of spring MVC we develop controller class by implementing spring specific interfaces or by extending from spring specific classes.This kills non invasive behavior of spring.
- It also makes the programmers to implements or override fixed methods with fixed signatures where programmer will be forced to placed unnecessarily parameters like req,res params even though it is not using them same thing can be said about return type of methods. most of the Controller classes are removed or deprecated in spring 4.x and spring 5.x in support of annotation,spring boot
- To develop controller classes as pojo classes having flexibility of placing user-defined methods with user-defined signatures we need to go for Annotations driven spring mvc. These annotations are introduced from **spring 2.5**. All new projects and in the migration projects annotations driven spring mvc programming is required.

Two important annotations are:-

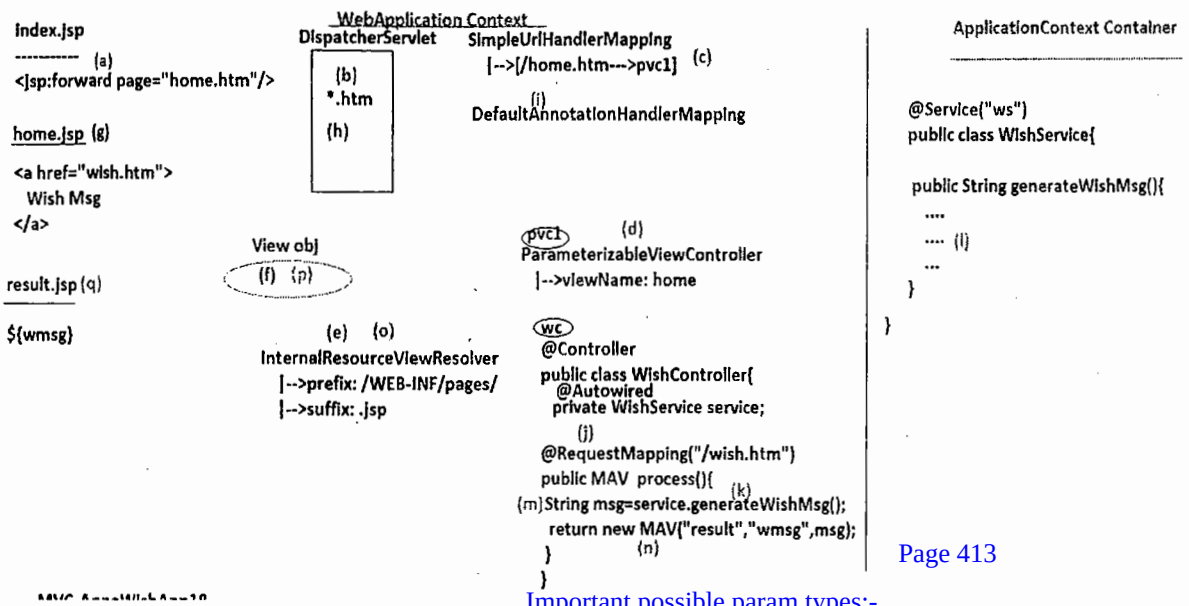
1) @Controller----->configure java class as spring mvc controller class and spring bean.
2) @RequestMapping---->To map incoming request urls with the methods of controller class based on the virtual path of request url.

- While working with annotations based spring mvc controllers, we can place flexible methods having flexible parameter types or flexible return types.
- eg: @Controller

Note:- @restController can be used as the alternate to @controller will working with spring webservices integration.

```
public class MyController
{
@RequestMapping ("/home.htm")
public <><method name>()
{
```

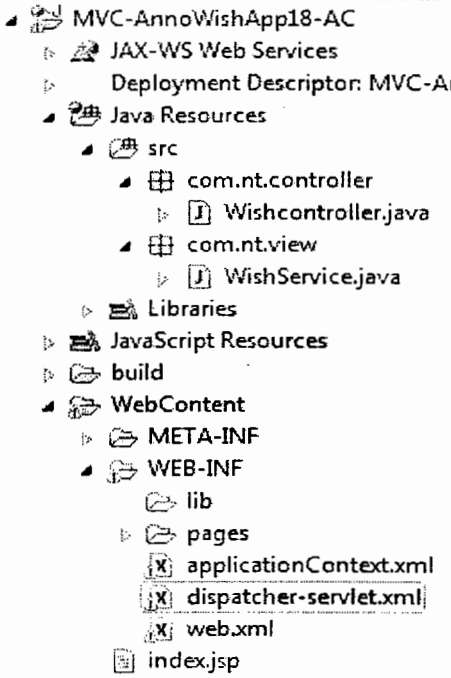
Diagram:



Page 413

Important possible param types:-
1-HttpServletRequest 2-HttpServletResponse 3-HttpSession
4-Command class 5-@ModelAtrribute 6-BindResult/Errors,map,mode,modelMap and etc
Note:-many other parameter types are there in support to restfull services integration
Important possible return type:- void,String,ModelAndView,Model,Map,View and etc
Many other return types are given in support to anotation
Note:-In spring Mvc only controller classes can be configure with annotations and the remaining component like HAndlerMappings,ViewResolver and etc must be configure throgh xmlfile.
AbstractController(using annotation):-
Source page:No form submission
Dest page:Static/dynamic form page with request processing
Note:- we still use built-in predefined controller like ParameterizableViewController using xml configuration".
Upto spring 2.5 we can work with only xml based spring mvc from 2.5 we can develop spring mvc app by mixing xml wit hannotation.
from spring 3.x we can develop spring mvc applicaion by avoiding xml including web.xml by using 100% java code approach

Limitation of using xml(programatic approach):-
1-here controller classes invassive(controller classes are tightly coupled with spring API)
2-Methods in controller classes can not be taken having flexible signature. they must be taken having fixed signature.
3-most of the controller classes are removed or deprecated in spring 4.x and spring 5.x in support of annotations.



Jars: same +AOP+mvc+javaxinject.jar

while working with annotation based spring mvc application configure predefine classes in xml file and configure userdefine classes throw annotation and scan them in xml file using contex:component-scan> tag.

WishService.java

```
package com.nt.service;
import java.util.Calendar;
import javax.inject.Named;
@Named("ws")
public class WishService {
    public String generateWishMsg(){
        // generate wish Msg based on current hour the day
        Calendar cl=Calendar.getInstance();
        int hour=cl.get(Calendar.HOUR_OF_DAY);
        if(hour<=12)
```

Q:-how the request trapped by dispatcher servlet executes the Handler method of controller classwhile working with annotation.
A:-Dispacher servlet traps the request-->handsover the request to defaultAnnotationHandlerMapping--->gets the packages from component:scan----->gets all the classes that are annotated with @controlller--->and searches fro the method that are annotated with @RequestMapping having incomming request url----->gets the signature of that handler method--->and calls that method with appropriat arguments

```
                return  "Good morning";
        else if(hour<=16)
                return "Good After noon";
        else if (hour<=20)
          return "Good Evening";
        else
                return "Good Night";
    }//method
}//class
```

WishController.java

```
package com.nt.controller;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import com.nt.service.WishService;
@Controller
public class WishController {

    @Resource
    private WishService service;

/*  @RequestMapping("/wish.htm")
    public  ModelAndView process(){
            //use service class
            String msg=service.generateWishMsg();

            return new ModelAndView("result","wmsg",msg);
    }//process */

/*  @RequestMapping("/wish.htm")
    public  String  process(Model model){

            //use service class
            String msg=service.generateWishMsg();

            // add data to mode
            model.addAttribute("wmsg",msg);

            return "result";
    } */
    /*      @RequestMapping("/wish.htm")
    public  String  process(ModelMap map){

            //use service class
            String msg=service.generateWishMsg();

            map.addAttribute("wmsg", msg);
```

```
            return "result";
        } */

    /*  @RequestMapping("/wish.htm")
        public  String  process(Map<String,Object> map){
```

best way of the writing handler method for abstract controller

```
            //use service class
            String msg=service.generateWishMsg();
```

reason are 1- total method is non spring api method
2-we have control on logical view name
3-dispatcherservlet itself creates map object that is required to place model attributes.

```
            map.put("wmsg",msg);
            return "result";
        } */

        @RequestMapping("/wish.htm")
        public void process(Model model){
            //Use service class
            String msg=service.generateWishMsg();

            model.addAttribute("wmsg",msg);
        }
}//class
```

Note:-while working with annotation based controller if handler method return type is void or map or model modelmap then the virtual path of incoming request url will be taken as logical view name.

index.jsp

```
<jsp:forward page="home.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/home.htm">pvc1</prop>
</props>
</property>
</bean>
<bean id="pvc1"
   class="org.springframework.web.servlet.mvc.ParameterizableViewController">
<property name="viewName"  value="home"/>
</bean>
<bean     class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotation
HandlerMappin    g"/>
<!-- <bean        class="org.springframework.web.servlet.view.DefaultRequestTo
ViewNameTranslator"/> -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
<context:component-scan base-package="com.nt.controller"/>
</beans>
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<context:component-scan base-package="com.nt.service"/>
</beans>
```

home.jsp

```
<h1><center> Welcome to Annotations Based Spring MVC</center></h1>
<a href="wish.htm"> wish Msg</a>
```

result.jsp

```
<center><h1> Result Page</h1></center>
<br>
Wish Message :  ${wmsg}
<br><a href="home.htm">home</a>
```

wish.jsp

```
<b> wish.jsp</b>
   Result is : ${wmsg}
```

default RequesToViewNameTranslator:

- If controller class handler() (annotated with @RequestingMapping) returntype is void (i.e not returning logical view name) then DispatcherServlet takes request uri (virtual path) of current request and translates into view name by trimming slashes and extension by using "org.springframework.servlet.view.RequestToViewNameTranslator"

In controller class:

```
@RequestMapping ("/wish.htm")
public void process (Model model)
{
.......................
.......................
}
```

- if the request url is http://localhost:3030/WishApp/wish.htm then "wish" will be taken as logical view name andwish.jspwill betaken as view Resource if the resolver is InternalResourceViewResolver. For this DispatcherServlet internally uses. DefaultRequestToViewNameTranslator". This translator strips/removes leading and trailing slashes and also remove extension from Request uri (makes "/home.htm" as "home") to use the result as logical viewname.

Annotation based AbstractCommandController:

Source Page:

- Form page with form submission (command class required)
- No serverside form validation
- No form handling.(displys same page with old values when form errors will come)

Destination Page:

- Static/dynamic page through Request processing.
- Use @ModelAttribute based param in handler method of controller class to specify the command class and its object to perform request wrapping (writing form data into command class object)
- @ModelAttribute is multipurpose annotation. It can be used to exposed any data to webview, to make parameter of handler method as,Command class to place the return value of handler method as ModelAttribute and etc...

Diagram:

Annotation based AbstractCommandController

Source Page:
- -> Form page with form submission (Command class is required)
- -> No Server Side form validations
- -> No Form handling

Dest page:
- -> Static/Dyamic page through request processing

Use @ModelAttribute based param in handler method of controller class to specify the Command class and its obj to perform request wrapping (Writing form data into Command class obj)

```
                web applicationContext                                                    applicationContext
                                          DispatcherServlet   SimpleUrlHandlerMapping
index.jsp                                                       |--->[ search.htm <-->pvc1]
--------------                                                        (c)
<jsp:forwa-- (a)                          (b)                   DefautlAnnotationHandlerMapping
   page="  search.htm">                   *.htm                        (j)
                                          (i)
                                                                pvc1    (d)
search.jsp  (g)                                                 ParameterizableViewController
                                                                  |---->viewName: search
<form action="performsearch.htm">
  No: <input type="text" name="no">          View object
  ...                                          (f) (p)          @Controller
  ...                                                           public class EmpSearachController{
  <input type="submit" value="search">  (h)                                  (k)
</form>                                                         @RequestMapping("/performsearch.htm")
                                  (e) (o)
                        InternalResourceviewResolver            public String search(ModelMap map,
                                 |--->prefix: /WEB-INF/pages/                  (m)      @ModelAttribute SearchCmd cmd)
                                 |--->suffix: .jsp
public class SearchCmd{                                         {                                Recongnizes SearchCmd as the
  private int no,salary;                                          .....                                   command class.
  private String name,desg;                                       .....
  // setters and getters                result.jsp (q)           .....
  .....     (l)-->request              ------------              return "result";
  ....          wrapping               display results                   (n)
}                                                               }
```

```
◢ MVC-AnnoEmpSearchApp19-ACC
  JAX-WS Web Services
  Deployment Descriptor: MVC-AnnoEmpSearchA
  ◢ Java Resources
    ◢ src
      ◢ com.nt.command
          SearchCmd.java
      ◢ com.nt.controller
          EmpSearchController.java
      ◢ com.nt.dto
          EmployeeDTO.java
    Libraries
  JavaScript Resources
  build
  ◢ WebContent
    META-INF
    ◢ WEB-INF
        lib
        pages
        applicationContext.xml
        dispatcher-servlet.xml
        web.xml
      index.jsp
```

SearchCmd.java

```
package com.nt.command;
public class SearchCmd {
        private int no,salary;
        private String name,desg;
        public int getNo() {
         return no;
        }
        public void setNo(int no) {
         this.no = no;
        }
        public int getSalary() {
         return salary;
        }
        public void setSalary(int salary) {
         this.salary = salary;
        }
        public String getName() {
         return name;
        }
        public void setName(String name) {
         this.name = name;
        }
        public String getDesg() {
         return desg;
        }
        public void setDesg(String desg) {
         this.desg = desg;
        }
}
```

EmployeeDTO.java

```
package com.nt.dto;
import java.io.Serializable;
public class EmployeeDTO implements Serializable {
        private int no,salary;
        private String name,desg;
        public EmployeeDTO() {
         // TODO Auto-generated constructor stub
        }
        public EmployeeDTO(int no, int salary, String name, String desg) {
         this.no = no;
         this.salary = salary;
         this.name = name;
         this.desg = desg;
        }
        public int getNo() {
         return no;
        }
        public void setNo(int no) {
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/search.htm">pvc1</prop>
</props>
</property>
</bean>
<bean id="pvc1" class="org.springframework.web.servlet.mvc.Parameterizable
ViewController">
<property name="viewName"  value="search"/>
</bean>
<bean  class="org.springframework.web.servlet.mvc.annotation.
DefaultAnnotationHandlerMapping"/>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
```

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

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
</beans>
```

emplist.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>
<center><h1>Search Results</h1></center>
<c:if test="${not empty searchResultsList}">
<table>
<tr>
<th>Emp No</th>
<th>Emp Name</th>
<th>Salary</th>
<th>Desg</th>
</tr>
<c:forEach items="${searchResultsList}"  var="result">
<tr>
<td><c:out value="${result.no}"/></td>
<td><c:out value="${result.name}"/></td>
<td><c:out value="${result.salary}"/></td>
<td><c:out value="${result.desg}"/></td>
</tr>
</c:forEach>
</table>
</c:if>
```

search.jsp

```
<h1><center> Emp Search Page</center></h1>
<br>
<form action="performsearch.htm" method="post">
  No :  <input type="text" name="no"><br>
  Name :  <input type="text" name="name"><br>
  Salary :  <input type="text" name="salary"><br>
  Desg :  <input type="text" name="desg"><br>
<input type="submit"  value="search">
</form>
```

AnnotationBased SimpleformController (or) AnnotationDrivenSimpleFormController:

Source Page:

Form page-->form submission
Form validation (server side)
Form handling

Destination Page:

Static/Dynamic page that comes after request process

InitialPhase Request (reqmethod=GET)(withrespect to xml)

Create command object

init Binder ()

add command object to model attribute

referencedata() and etc...

PostBack request (req method=POST)(with respect to xml)

Create/locate command object

initBindier ()

request wrapping

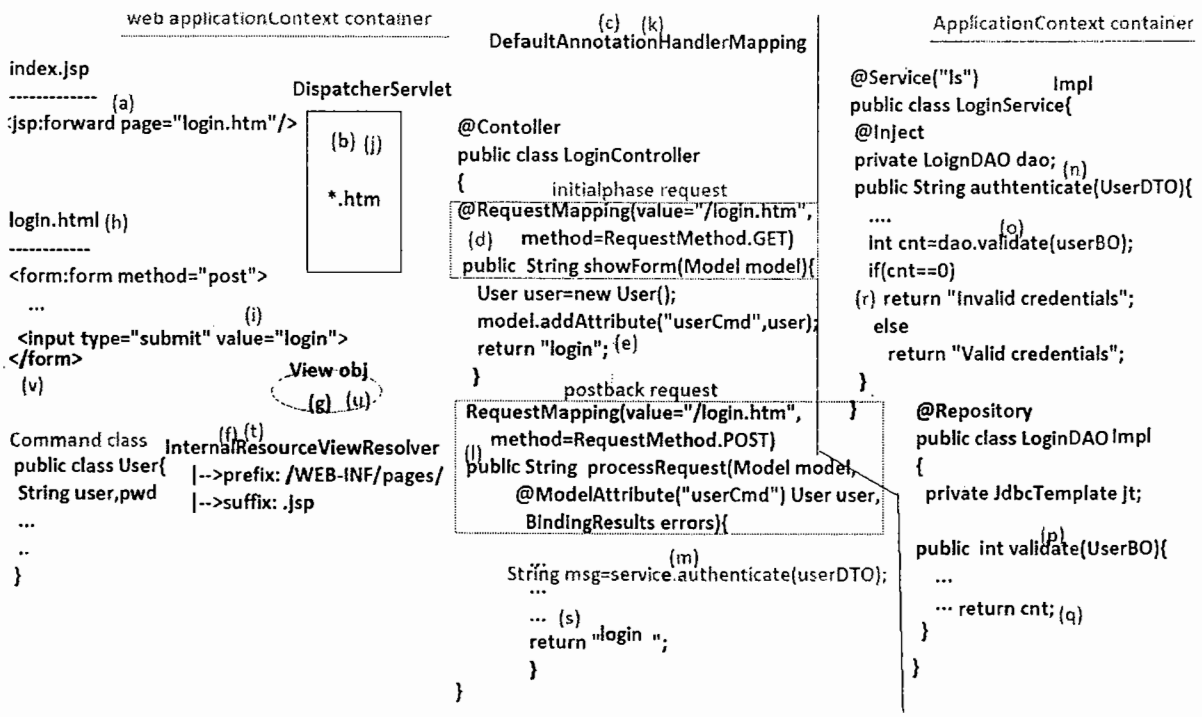
BindException object creation

Validation support (-), validate (-)

Referencedata

onSubmit() and etc....

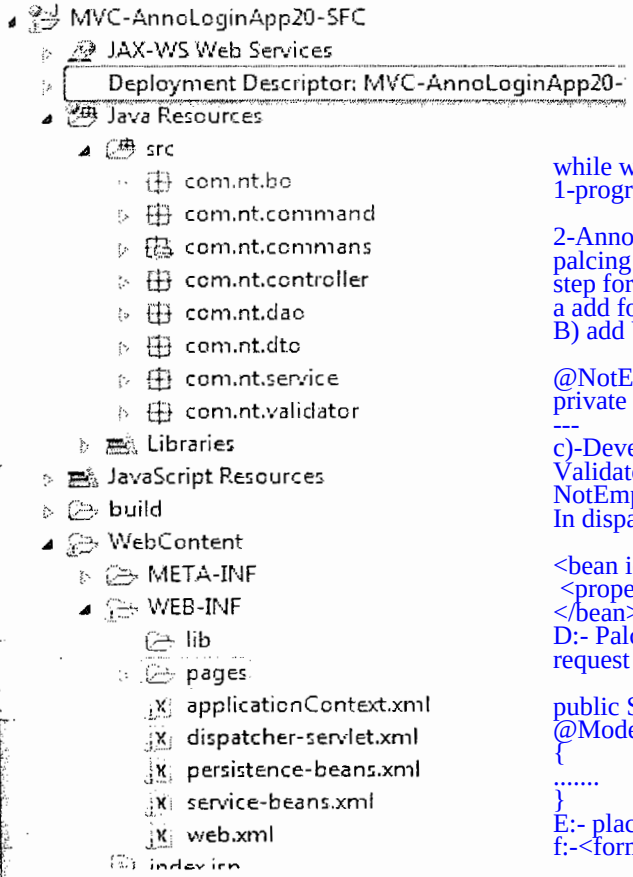
Diagram



Sample AnnotationBasedSimpleFormController Class

```
@Controller
public class LoginController
{
Initial phase request
@RequestMapping (value="/login.hmt", method=RequestMethod.GET)
public String showForm(Model model){
User user=new User();
model.addAttribute("usercmd", user);
return "login";
}
@RequestMapping (value="/login.htm", method=RequestMethod.POST)
```

```
    public String processRequest(Model model,@ModelAttribute ("usercmd") User user,
        BindingResult errors){
    String msg=service.authenticateconfigure:userDTO);
    ....................
    ....................
    return "login";
}
}
```



while working with annotation based mvc we can do form validation in 2 ways
1-programatic approach(Developing Validator class implementing Validator)

2-Annotation approach(Developing command class with validator annotation and by palcing JAvaEE , HibernateValidator Annotation in command class)
step for Annotaion approach
a add following <dependency> tag in pom.xml
B) add Validation Annotation Command class properties

@NotEmpty(message="user is mandatory")
private Strring user;
---
c)-Develop properties file having validation error message and also cfg that properties file
Validator properties:-
NotEmpty.userCmd.user(property name)=Username is required
In dispatcher-servlet.xml:--

<bean id="messageSource" class="pkg.ResourceBundleMessageSource">
<property name="basename" value="com/nt/commons/validator"/>
</bean>
D:- Palce @valid Annotation on parameter of Command class that process Post back request in controller class

public String processForm(Map<...,...>map,@valid
@ModelAttribute("userCmd")UserCommand)
{
.....
}
E:- place <mvc : annotation-driver> tag in dispacher-servlet.xml
f:-<form : errors >in login.jsp under <form:form >tag

UserBO.java

```
package com.nt.bo;
public class UserBO {
    private String user,pwd;

    public String getUser() {
            return user;
    }
    public void setUser(String user) {
            this.user = user;
    }
    public String getPwd() {
            return pwd;
    }
    public void setPwd(String pwd) {
            this.pwd = pwd;
    }
}
```

LoginDao.java

```
package com.nt.dao;
import com.nt.bo.UserBO;
public interface LoginDao {
    public int validate(UserBO ubo);
}
```

LoginDaoImpl.java

```
package com.nt.dao;
import javax.inject.Inject;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import com.nt.bo.UserBO;
@Repository
public class LoginDaoImpl implements LoginDao {
    private static final String AUTH_QRY="SELECT COUNT(*) FROM USERLIST WHERE
UNAME=?     AND PWD=? ";
    @Inject
    private JdbcTemplate jt;
    @Override
    public int validate(UserBO ubo) {
     int cnt=jt.queryForInt(AUTH_QRY, ubo.getUser(),ubo.getPwd());
            return cnt;
    }
}
```

UserDTO.java

```
package com.nt.dto;
import java.io.Serializable;
public class UserDTO implements Serializable {
    private String user,pwd;
    public String getUser() {
            return user;
    }
    public void setUser(String user) {
            this.user = user;
    }
    public String getPwd() {
            return pwd;
    }
    public void setPwd(String pwd) {
            this.pwd = pwd;
    }
}
```

LoginService.java

```
package com.nt.service;
import com.nt.dto.UserDTO;
public interface LoginService {
    public String authtenticate(UserDTO udto);
}
```

LoginServiceImpl.java

```
package com.nt.service;
import javax.inject.Inject;
import org.springframework.stereotype.Service;
import com.nt.bo.UserBO;
import com.nt.dao.LoginDao;
import com.nt.dto.UserDTO;
@Service
public class LoginServiceImpl implements LoginService {
    @Inject
    private LoginDao  dao;
    @Override
    public String authtenticate(UserDTO udto) {
            //convert DTO object to BO object
            UserBO ubo=new UserBO();
            ubo.setUser(udto.getUser());
            ubo.setPwd(udto.getPwd());
            //Use DAO class
            int cnt=dao.validate(ubo);
            if(cnt==0)
                    return "InValid Credentials";
            else
                    return "Valid Credentials";
    }
}
```

LoginValidator.java

```
package com.nt.validator;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.nt.command.User;
@Component
public class LoginValidator implements Validator {
    @Override
    public boolean supports(Class<?> clazz) {
            return clazz.isAssignableFrom(User.class);
    }
    @Override
    public void validate(Object cmd, Errors errors) {
            //type casting
            User ucmd=(User)cmd;
            if(ucmd.getUser()==null || ucmd.getUser().equals("") ||
    ucmd.getUser().length()==0){
                    errors.rejectValue("user", "user.required");
```

```
            }

            if(ucmd.getPwd()==null || ucmd.getPwd().equals("") ||
        ucmd.getPwd().length()==0){
                    errors.rejectValue("pwd","pwd.required");
            }
        }}
```

Errors.properties

```
user.required=Username is mandatory
pwd.required=Password is mandatory
```

User.java

```
package com.nt.command;
import java.util.Date;
public class User {
    private String user,pwd,dmn;
    private Date dor;
    public String getUser() {
            return user;
    }
    public void setUser(String user) {
            this.user = user;
    }
    public String getPwd() {
            return pwd;
    }
    public void setPwd(String pwd) {
            this.pwd = pwd;
    }
    public String getDmn() {
            return dmn;
    }
    public void setDmn(String dmn) {
            this.dmn = dmn;
    }
    public Date getDor() {
            return dor;
    }
    public void setDor(Date dor) {
            this.dor = dor;
    }
}
```

Xml based Simple form Controller	Annotation based Simple formController
onSubmit(-,-,-,-)	@RequestMapping(---,method=POST) public String ShowForm Attribute(?) (command class)
formBacking object	@RequestMapping(---,method=GET) public String ShowForm (Mapping){---}
Map refereceData(-,-)	@ModelAttribute(---) public List<--->populateDomains(){--}

void initBinder(ServletRequest Binder){
------- 	
}
	@InitBinder
	public void myInitBinder(WebDataBinder binder){-----
	}

handle InvalidSubmmit()	no direct method place extra logic in Hander Intercepter

LoginController.java

```
package com.nt.controller;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

```
import javax.inject.Inject;
import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.nt.command.User;
import com.nt.dto.UserDTO;
import com.nt.service.LoginService;
import com.nt.validator.LoginValidator;
@Controller
public class LoginController {
    @Inject
    private LoginService service;
    @Inject
    private LoginValidator validator;
    @RequestMapping(value="/login.htm",method=RequestMethod.GET)
    public String showForm(Model model){
            //create command object and keep it as model attribute
            User user=new User();
            model.addAttribute("userCmd",user);
            //return lvn
            return "login";
    }
    @RequestMapping(value="/login.htm",method=RequestMethod.POST)
    public String  processForm(Model model,@ModelAttribute("userCmd")User
    cmd,BindingResult errors){
            //convert Command object to DTO object
            UserDTO udto=new UserDTO();
            udto.setUser(cmd.getUser());
            udto.setPwd(cmd.getPwd());

            //use validator
            if(validator.supports(User.class)){
                    validator.validate(cmd, errors);
                      if(errors.hasErrors()){
                              return "login";
                      }
            }
            //use service class
            String result=service.authtenticate(udto);
            model.addAttribute("result",result);
            //return lvn
            return "login";
    }
```

```
        @ModelAttribute("domains")  // alternate for referenceData
        public List<String>  populateDomains(){
                List <String>domainsList =new ArrayList<String>();
                 domainsList.add("Gmail.com");
                 domainsList.add("yahoo.com");
                 domainsList.add("rediff.com");
                 return domainsList;
        }
        @InitBinder
        public void  myInitBinder(WebDataBinder binder){
                SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
                binder.registerCustomEditor(Date.class,
                                                        new CustomDateEditor(sdf,true));
        }
    }
```

index.jsp

```
    <jsp:forward page="login.htm"/>
```

web.xml

```
    <web-app>
    <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
</web-app>
```

service-beans.xml

```
    <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
```

```
http://www.springframework.org/schema/context/spring-context.xsd">
<context:component-scan base-package="com.nt.service"/>
</beans>
```

persistence-beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
<context:component-scan base-package="com.nt.dao"/>
</beans>
```

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean      class="org.springframework.web.servlet.mvc.annotation.Default
AnnotationHandlerMapping"/>
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundle
MessageSource">
<property name="basename" value="com/nt/commons/errors"/>
</bean>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
<context:component-scan base-package="com.nt.controller,com.nt.validator"/>
</beans>
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<import resource="service-beans.xml"/>
<import resource="persistence-beans.xml"/>
</beans>
```

login.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h1><center>  Login Page</center></h1>
<form:form method="post" commandName="userCmd">
<span style='color:red'><form:errors path="*"/></span><br>
  Username: <form:input path="user"/><br>
  Password : <form:password path="pwd"/><br>
  Domains: <form:checkboxes items="${domains}" path="dmn"/><br>
 Date of Registration: <form:input path="dor"/>(dd-MM-yyyy) <br>
<input type="submit"  value="Login"/><br>
</form:form>
<br>
<c:if test="${result ne null}">
<c:out value="${result}"/>
</c:if>
<c:out value="${userCmd.dor }" />
```

⇨ **To perform form validation while working with SimpleFormController with annotations:-**

1) Prepare properties file having form validation error messages.
2) Configure properties file.
3) Develop validator class implementing Validator (i) and overriding supports(-), validate(-) methods.
4) Inject validator class in controller class and call supports(-), validate(-) methods in the handler method, that can process postback request.
5) Place <form:errors> tag in form page to display form validation errors.

While working with annotation based SimpleFormController we can use @ModelAttribute annotation method as alternate to referenceData() of xml based SimpleFormController. This holds data that has to be displayed in the form page as additional data as part of initial phase request and postback request with validation error messages.

In controller class:

```
@ModelAttribute ("domains")
public List<Stirng> populateDomains()
{
List<String>domainList=new ArrayList <String> ();
```

```
domainList.add("Gmail.com");
domainList.add("Yahoo.com");
domainList.add("rediff.com");
return domainList();
}
```

In form Page:

- Domains :<form:checkboxes items="${domains}" path="dmn"/>

In order to register property editors that are required to convert form supplied String values to various types of command class properties we override InitBiner() in our controller class while working with xml configuration based SimpleFormController.We can bring the same affect by using @InitBinder while using annotations driven SimpleFormController.

- eg :

In Controller Class

```
@InitBinder
public void myInitBinder(WebDataBinder binder)
{
SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
    binder.registerCustomEditor(Date.class, new CustomDateEditor(sdf, true));
    }
```

File Uploading and file downloading:

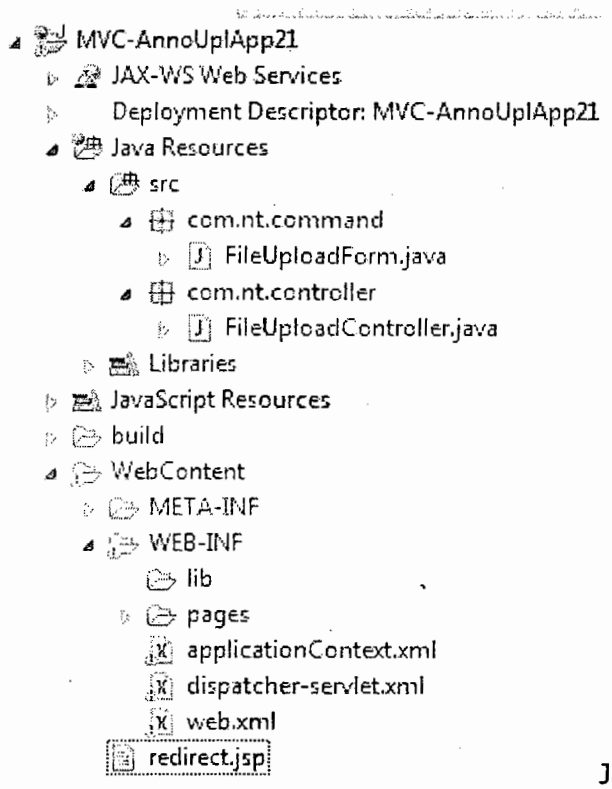
- The process of selecting file from client machine file system and sending that file to server system file system is called file uploading and reverse is called File downloading.
- While developing matrimony application, job portal, video sharing apps the file uploading and downloading are quietly required.
- Spring MVC gives built-in support for fileuploading.We need to configure "**CommonsMultipartResolver**" in dispatcher-servlet.xml to make controller class ready to resolve multi MIME TYPES/parts based form data (form data having uploading files)

Using this class we can also configure properties like "maxUploadSize", "maxInmemorySize", "defaultEncoding" properties.

Perform the following operations for file uploading:

1) Take file uploading component in form page and also specify "enctype"andrequest method as "POST".
2) Takes command class having propery of type "**MultipartFile**" to receive and hold uploaded file.
3) Configure "CommonsMultipartResolver" class as bean in dispatcher-servlet.xml
4) Write the stream based logic in controller to save the rececive in server machine file system.

if commans multipartResolver is not configure the controller that receives request from form submission does not recognise other than text data send by form so request wrapping into command object having uploaded files do not take place.to over come this problem we must configure commons mulipartresolver having fixed bean id multipart resolver



Jar: MVC jarc, AOP jars,
commons-fileupload-1.1.1.jars,
commons-io-1.3.1.jars.

rediredct.jsp

```
<jsp:forward page="home.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
<welcome-file-list>
<welcome-file>redirect.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<bean     class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotation
HandlerMapping"/>
<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
<property name="suffix"  value=".jsp"/>
</bean>
<context:component-scan base-package="com.nt.controller"/>
</beans>
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

file_upload_form.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
    <h2>File Upload Application</h2>
    <form:form method="post" action="save.htm" modelAttribute="uploadForm"
    enctype="multipart/form-data">
            Please select a file to upload : <input type="file" name="file" /><br/>
            <input type="submit" value="upload" />
    </form:form>
</body>
</html>
```

file_upload_success.jsp

```
    ${files} uploaded successfully.
```

FileUploadController.java

```
package com.nt.controller;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.multipart.MultipartFile;
import com.nt.command.FileUploadForm;
@Controller
public class FileUploadController {
    @RequestMapping(value = "/home.htm",
                                        method = RequestMethod.GET)
    public String displayForm(Model model) {
            return "file_upload_form"; //returns the page name to ViewResolver ultimately
    }
    @RequestMapping(value = "/save.htm", method = RequestMethod.POST)
    public String upload(Model map,@ModelAttribute("uploadForm")
FileUploadForm cmd) {
     File dir=new File("E:/springfileuploads/");
            //get Uploaded file
            MultipartFile file = cmd.getFile();
            String fileName = "";
            if (file != null) {
                    fileName = file.getOriginalFilename();
                    System.out.println("orginal file name="+fileName);
                    //Handle file content - multipartFile.getInputStream()
                    InputStream inputStream=null;
                    OutputStream outputStream;
                    try {
                            inputStream = file.getInputStream();
                     //If file is not present then only create a file and write the data into
    them
                            if(!dir.exists()){
                                    dir.mkdir();
                                    }
                            File newFile = new File("E:/springfileUploads/"+fileName);
                          outputStream = new FileOutputStream(newFile);
                            int read = 0;
                                    byte[] buffer = new byte[1024];
                    while ((read = inputStream.read(buffer)) != -1) {

                                            outputStream.write(buffer, 0, read);
                                    }
                                    outputStream.close();
                                    inputStream.close();
                    } catch (IOException e) {
                                    System.out.println("Exception "+e);
                    }
            }
            map.addAttribute("files", fileName);
            return "file_upload_success";
    }
}
```

FileUploadForm.java

```
package com.nt.command;
import org.springframework.web.multipart.MultipartFile;
public class FileUploadForm {
    //Can be used to hold the uploaded file
    private MultipartFile file;
    public MultipartFile getFile() {
            return file;
    }
public void setFile(MultipartFile file) {
            this.file = file;
    }
}
```

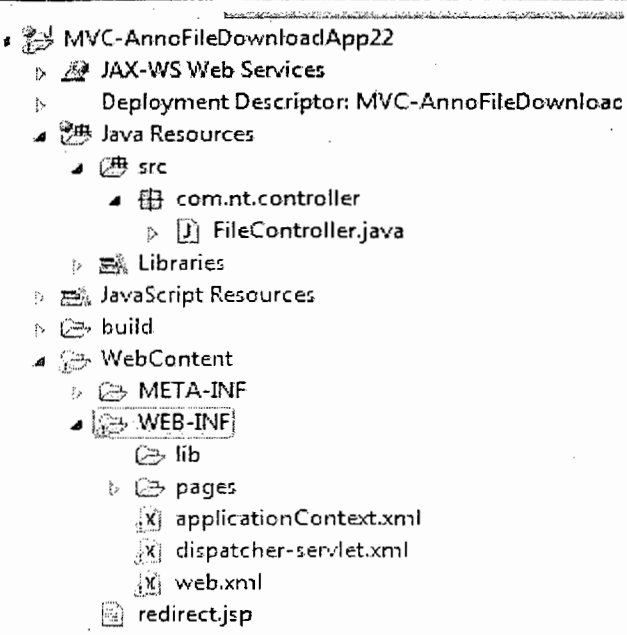
Downloading:

- The process of getting the file belonging to server machine file system to client machine file system is called file downloading. By using "**Content-Disposition**" header we can give instruction to browser to display the generated output as downloadable or attachable file.

For example application on file uploading.
We can use @RequestParam annotation to store the received request parameter value into handlermethod parameter.

To download file we need to do following activities in the controller class handler method:

1) Locate the file.
2) Get the length of the file and set it as response content length.
3) Get the MIME type of the file and set it as response content type.
4) Get input stream pointing to the file.
5) Get output stream pointing to the response object.
6) Use buffer based logic to read file content and to write to response object.
7) Specify content-disposition header having attachment of the value to make browser to display the received content as the downlodable file.



MVC library Jars:-

commons-fileupload-1.1.1.jar

commons-io-1.3.1.jar

index.jsp

```
<jsp:forward page="home.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

dispatcher-servlet.xml

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

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

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

    http://www.springframework.org/schema/context/spring-context.xsd
                                    http://www.springframework.org/schema/mvc
                    http://www.springframework.org/schema/mvc/spring-mvc.xsd">
Since No handler mapping is configured the default AnnotationHandlerMapping will
    take place.
<context:component-scan base-package="com.nat.controller"/><!—
package name where controllers are present -->
<context:annotation-config />
<!-- View Resolver,pointing to WEB-INF/jsp folder -->
<bean id="jspViewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView" />
<property name="prefix" value="/WEB-INF/pages/" />
<property name="suffix" value=".jsp" />
</bean>
</beans>
```

file_download_form.jsp

```
<%@page import="java.io.File" %>
<table border=1>
<tr><td>FileName</td><td>Download</td></tr>
<%  String path = "E:/springfileuploads";
        String files;
        File folder = new File(path);
        File[] listOfFiles = folder.listFiles();

        for (int i = 0; i < listOfFiles.length; i++)
        {
         if (listOfFiles[i].isFile())
         {
         files = listOfFiles[i].getName();
%>
<tr><td><%= files%></td>
<td><a href="download.htm?file_name=<%= files%>">Download</a></td></tr>
<% }
        }
        %>
</table>
</html>
```

FileDownloadController.java

```
package com.nat.controller;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.io.IOUtils;
import org.springframework.core.io.FileSystemResource;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
@Controller
public class FileDownloadController {
    @RequestMapping(value = "/home.htm")
    public String displayForm() {
            return "file_download_form"; //returns the page name to
                                                    ViewResolver ultimately
    }
    @RequestMapping(value = "/download.htm")
    public void  download(@RequestParam("file_name") String fileName,
                                                    HttpServletResponse        response)
{
            InputStream is=null;
            try {
            System.out.println("Requested File"+fileName+" response="+response);
            String filePath="E:/springfileuploads/"+fileName;
        // get your file as InputStream
        is=new FileInputStream(filePath);

        //Remove the below code,will dipatch the file in view mode like txt,bat file
            response.setHeader("Content-Disposition", "attachment; filename=
    "+fileName);
            IOUtils.copy(is, response.getOutputStream());
        response.flushBuffer();

      } catch (IOException eg) {
        //log.info("Error writing file to output stream. Filename was '" + fileName + "'");
        throw new RuntimeException("IOError writing file to output stream");
      }
    }
}
```

SpringMVC with Tiles:

A tile is a logical portion / partician in a webpage like header tile, footer tile, body tile and etc.....

Appache tiles framework/plugin is given to construct webpages based on layout having layout control. This is like working with compositeView design pattern.

Layout control means all the web pages will be constructed based on single layout page for uniform look and any modification in layout page will be reflected to all webpages popular layouts are:-

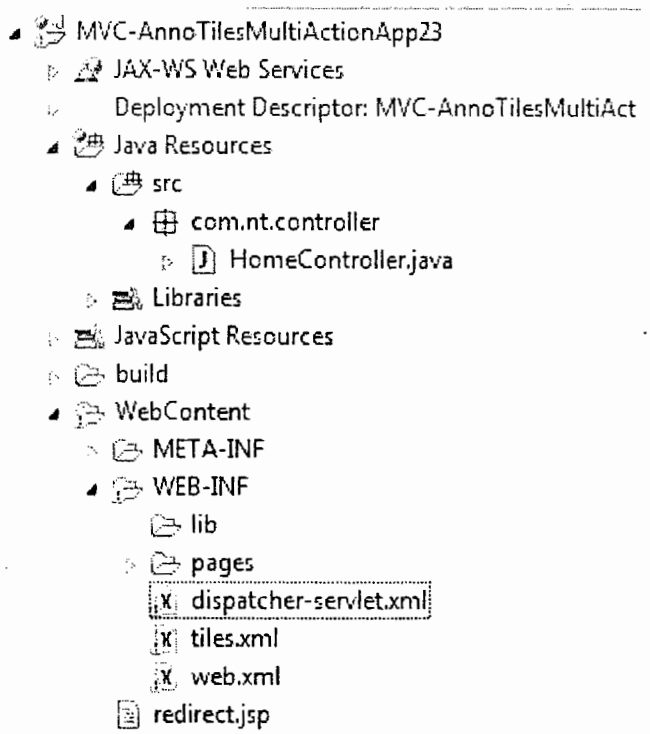
a) Classic layout
b) Two-Column layout
c) Three-Column layout
d) Circular layout

⇨ Spring gives built-in support to work with tilesFor this we need to configure "TileConfigurer" specifying the tile definition file name(xml). The default file name is /WEB-INF/tiles.xml.

⇨ Every tile definition specifies the tile values i.e each webpage content based on layout page. We create tile definition based on 1 per webpage basis by having logical name. we can use "FilesViewResolver" to display webpage based on each tile definition.

Procedure to develop spring MVC with tiles application:

1) Design layout page for templating webpages and to maintain webpages having layout control.
2) Develop files xml in WEB-INF folder having tile definitions based on no of webpages in our website on tile definition for webpage basis. We can use tile definition inheritance concept here.
3) Configure "TilesConfigurer" class in dispatcher –servlet.xml file as spring bean specifying "tile definition configuration " filename.
4) Configure "TilesViewResolver" as ViewResolver.
5) Make acontroller class handler methods to return "tile definition name" as logical view name.

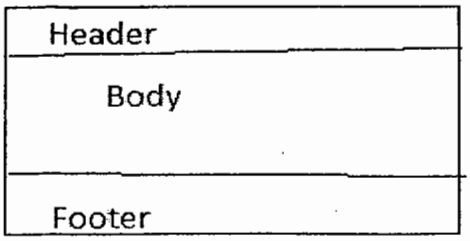


Jar:- MVC jarc, AOP jar,Tiles-template-2.2.2.jar,Tiles-servlet, tiles-core, tiles-jsp, tiles-api,

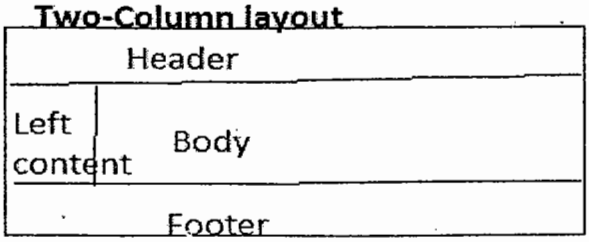
SL4j-api, SL4j-log4j12, JSTL, antlr.jar

Commons-beanutils, commons-collections, commons-deigester, commons-fileupload, commons-10-jar.

Classic layout



Two-Column layout



(f)

HomeController.java

```
package com.nat.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class HomeController {
    @RequestMapping("/demo") (p)
(f)
    public ModelAndView hello() {
return new ModelAndView("page1", "message", "Body Content goes Here !");
(g)
    }
                    tile definition with lvn

    @RequestMapping("/link1")
    public ModelAndView register() {
  return new ModelAndView("page2", "message", "Registration page");
(q)
    }
    @RequestMapping("/link2")
    public ModelAndView login() {
return new ModelAndView("page3", "message", "Login Page");
    }
    @RequestMapping("/link3")
    public ModelAndView reset() {
 return new ModelAndView("page4", "message", "Login Page");
    }
}
```

(b) ***redirect.jsp*** (c)

```
<jsp:forward page="demo.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<welcome-file-list>
(a)
<welcome-file>redirect.jsp</welcome-file>
</welcome-file-list>
(d) <servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
    </servlet>
(N) <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
(*d) </servlet-mapping>
</web-app>
```

tiles.xml

```
    <!DOCTYPE tiles-definitions PUBLIC
            "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
            "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
    <tiles-definitions>
    <definition name="baseDef" template="/WEB-INF/pages/Layout.jsp">
    <put-attribute name="title" value="" />
    <put-attribute name="header" value="/WEB-INF/pages/Header.jsp" />
    <put-attribute name="menu" value="/WEB-INF/pages/Menu.jsp" />
    <put-attribute name="body" value="" />
    <put-attribute name="footer" value="/WEB-INF/pages/Footer.jsp" />
    </definition>
(k) <definition name="page1" extends=" baseDef"">
    <put-attribute name="title" value="Home Page" />
    <put-attribute name="body" value="/WEB-INF/pages/Hello.jsp" />
    </definition>
    <definition name="page2" extends=" baseDef"">
(u) <put-attribute name="title" value="course page" />
    <put-attribute name="body" value="/WEB-INF/pages/courses.jsp" />
    </definition>
    <definition name="page3" extends=" baseDef"">
    <put-attribute name="title" value="faculty page" />
    <put-attribute name="body" value="/WEB-INF/pages/faculties.jsp" />
    </definition>
    <definition name="page4" extends="base.definition">
    <put-attribute name="title" value="address page"/>
    <put-attribute name="body" value="/WEB-INF/pages/address.jsp" />
    </definition>
    </tiles-definitions>
```

dispatcher-servlet.xml

```
    <beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
            since No handler Mapping is configured DefaultAnnotationHandlerMapping will
(e)         be taken (o)
        <context:component-scan base-package="com.nat.controller" />
```

```
    <bean id="viewResolver"(h) (r)
            class="org.springframework.web.servlet.view.UrlBasedViewResolver">
            <property name="viewClass"
    value="org.springframework.web.servlet.view.tiles2.TilesView"/>
</bean>
            <bean id="tilesConfigurer"
            class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
(J)         <property name="definitions">
(t)                 <list>
                        <value>/WEB-INF/tiles.xml</value>
                    </list>
            </property>
    </bean>
    <!--    <bean id="tilesviewResolver"
    class="org.springframework.web.servlet.view.tiles3.TilesViewResolver"/>
<bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
<property name="definitions">
<list>
<value>/WEB-INF/tiles.xml</value>
</list>
</property>
</bean>
 -->
</beans>
```

TilesView class obj

(i) (s)

Address.sjp

```
Naresh IT
 Satyam Theater,
Ameerpet
```

Courses.jsp

```
java
.net
Oracle
```

faculity.jsp

```
Hari Krishna
Satish Gupta
Vijay kumar
```

Footer.jsp

```
<center>
<p>Copyright &copy; www.nataraz.in</p>
</center>
```

Header.jsp

```
<marquee><b>Welcome to Tiles Example</b></marquee>
```

Hello.jsp

```
<%@page isELIgnored="false" %>
<center>${message}</center>
```

(l) ***layout.jsp*** (two clolumn layout) displays the web page through layout.jsp having the following tile value

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<html>
<head>
<title><tiles:insertAttribute name="title" ignore="true" />
</title>
</head>
<body>
<table border="1" cellspacing="2" align="center" width="100%" height="100%">
<tr bgcolor="cyan">
<td height="50" colspan="2"><tiles:insertAttribute name="header" />
</td>
</tr>
<tr>
<td height="250" width="150"><tiles:insertAttribute name="menu" />
</td>
<td width="400"><tiles:insertAttribute name="body" />
</td>
</tr>
<tr bgcolor='red'>
<td height="30" colspan="2"><tiles:insertAttribute name="footer" />
</td>
</tr>
</table>
</body>
</html>
```

title-->Home page
header-->header.jsp
menu-->menu.jsp
footer-->footer.jsp
body-->hello.jsp

(v)--displays the web page through layout.jsp having the following tile value
title--

menu.jsp

(m)

```
<a href="link1.htm">Courses</a><br>
<a href="link2.htm">faculties</a><br>
<a href="link3.htm">addresses</a>
```

Handling exceptions

HandlerExceptionResolver

Spring HandlerExceptionResolver implementations deal with unexpected exceptions that occur during controller execution. A HandlerExceptionResolversomewhat resembles the exception mappings you can define in the web application descriptor web.xml. However, they provide a more flexible way to do so. For example they provide information about which handler was executing when the exception was thrown. Furthermore, a programmatic way of handling exceptions gives you more options for responding appropriately before the request is forwarded to another URL (the same end result as when you use the Servlet specific exception mappings).

- Besides implementing the HandlerExceptionResolver interface, which is only a matter of

implementing the resolveException(Exception, Handler) method and returning a ModelAndView, you may also use the provided SimpleMappingExceptionResolver or create @ExceptionHandler methods. TheSimpleMappingExceptionResolver enables you to take the class name of any exception that might be thrown and map it to a view name. This is functionally equivalent to the exception mapping feature from the Servlet API, but it is also possible to implement more finely grained mappings of exceptions from different handlers. The @ExceptionHandler annotation on the other hand can be used on methods that should be invoked to handle an exception. Such methods may be defined locally within an @Controller or may apply to many @Controller classes when defined within an @ControllerAdvice class. The following sections explain this in more detail.

@ExceptionHandler

- The HandlerExceptionResolver interface and the SimpleMappingExceptionResolver implementations allow you to map Exceptions to specific views declaratively along with some optional Java logic before forwarding to those views.However, in some cases, especially when relying on @ResponseBody methods rather than on view resolution, it may be more convenient to directly set the status of the response and optionally write error content to the body of the response.

- You can do that with @ExceptionHandler methods. When declared within a controller such methods apply to exceptions raised by @RequestMapping methods of that contoroller (or any of its sub-classes). You can also declare an @ExceptionHandler method within an @ControllerAdvice class in which case it handles exceptions from @RequestMapping methods from many controllers. Below is an example of a controller-local @ExceptionHandler method:

```
@Controller
publicclass SimpleController {

// @RequestMapping methods omitted ...

@ExceptionHandler(IOException.class)
public ResponseEntity<String> handleIOException(IOException ex) {
// prepare responseEntity
return responseEntity;
  }}
```

- The @ExceptionHandler value can be set to an array of Exception types. If an exception is thrown that matches one of the types in the list, then the method annotated with the matching @ExceptionHandler will be invoked. If the annotation value is not set then the exception types listed as method arguments are used.

- Much like standard controller methods annotated with a @RequestMapping annotation, the method arguments and return values of @ExceptionHandler methods can be flexible. For example, the HttpServletRequest can be accessed in Servlet environments and the PortletRequest in Portlet environments. The return type can be a String, which is interpreted as a view name, a ModelAndView object, a ResponseEntity, or you can also add

the @ResponseBody to have the method return value converted with message converters and written to the response stream.

Handling Standard Spring MVC Exceptions

- Spring MVC may raise a number of exceptions while processing a request. The SimpleMappingExceptionResolver can easily map any exception to a default error view as needed. However, when working with clients that interpret responses in an automated way you will want to set specific status code on the response. Depending on the exception raised the status code may indicate a client error (4xx) or a server error (5xx). The DefaultHandlerExceptionResolver works transparently by setting the status of the response. However, it stops short of writing any error content to the body of the response while your application may need to add developer-friendly content to every error response for example when providing a REST API. You can prepare aModelAndView and render error content through view resolution — i.e. by configuring a ContentNegotiatingViewResolver, MappingJackson2JsonView, and so on. However, you may prefer to use @ExceptionHandler methods instead.

- If you prefer to write error content via @ExceptionHandler methods you can extend ResponseEntityExceptionHandler instead. This is a convenient base for@ControllerAdvice classes providing an @ExceptionHandler method to handle standard Spring MVC exceptions and return ResponseEntity. That allows you to customize the response and write error content with message converters. See the ResponseEntityExceptionHandler javadocs for more details.

Customizing the Default Servlet Container Error Page

- When the status of the response is set to an error status code and the body of the response is empty, Servlet containers commonly render an HTML formatted error page. To customize the default error page of the container, you can declare an <error-page> element in web.xml. Up until Servlet 3, that element had to be mapped to a specific status code or exception type. Starting with Servlet 3 an error page does not need to be mapped, which effectively means the specified location customizes the default Servlet container error page.

```
<error-page>

<location>/error</location>

</error-page>
```

Note that the actual location for the error page can be a JSP page or some other URL within the container including one handled through an @Controller method:

When writing error information, the status code and the error message set on the HttpServletResponse can be accessed through request attributes in a controller:

```
@Controller
publicclass ErrorController {
```

```
@RequestMapping(path = "/error", produces =
MediaType.APPLICATION_JSON_UTF8_VALUE)
@ResponseBody
public Map<String, Object> handle(HttpServletRequest request) {

        Map<String, Object> map = new HashMap<String, Object>();
        map.put("status", request.getAttribute("javax.servlet.error.status_code"));
        map.put("reason", request.getAttribute("javax.servlet.error.message"));

return map;
    }

}
```

or in a JSP:

```
<%@pagecontentType="application/json"pageEncoding="UTF-8"%>
{
status:<%=request.getAttribute("javax.servlet.error.status_code")%>,
reason:<%=request.getAttribute("javax.servlet.error.message")%>
}
```

Servlet dynamic registration from version 3.0::--

ServletRegistration.Dynamic register=sc.addServlet("dispatcher",new DispatcherServlet());
register.addMapping("*.htm");
register.setLoadOnStartUp(2);
while working with 100% code based application registration of predefine servlet component is not possible directly. beacause it will not come with @webservlet annotation and we can not use web.xml.
Dynamic servletListener registration:-
sc.addListener(new ContextLoaderListener());

how dynamic servlet registration take place servlet 3.x and 4.envirenment
propramer deploys the webapplication-container creates servlet-- service provider class loader searches for javax.servlet.ServletContainerInitializer file in muliple places of webapplication including in the jar file add to web-inf/lib folder form that file it loads the given servlet container initializer class---instantiate the class and calls on startup methods on that class having servletcontext object---this method contents logic to register the servlet component dynamicaly.

100% Java Code Based Spring MVC

Few months ago I started playing with Spring Framework, which I really enjoy. Looking for the tutorials I found some really great resources on the web as well as books. Unfortunately many of them were written for Spring 2.x, which lacks some recent features, mainly ability to configure everything by Java annotations (a.k.a. JavaConfig) rather than XML. That was something that kept me away from Spring at the first place - doing it programmatically gives much more sense of control and is more readable. So, that's what I'm trying to achieve here, I'll try to show you how to do it in more 'modern' way, showing integrations with various software packages along the way.

Let's start with a simple skeleton Spring MVC application. Since version 3.1 it has Servlet 3 API support, I won't use web.xml for configuring DispatcherServlet, rather I'll configure it programmatically.

Full source code for this article is here on GitHub

Maven configuration

Most important dependencies for this project will be spring-webmvc, which provides DispatcherServlet and pulls other Spring artifacts together with it and Servlet 3.0 API.

```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>3.2.4</version>
</dependency>
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>javax.servlet-api</artifactId>
<version>3.0.1</version>
</dependency>
```

Initializer

AppInitializer is a class that implements WebApplicationInitializer interface. We hook up to onStartup() method, to add DispatcherServlet to ServletContext.

```
publicclassAppInitializerimplementsWebApplicationInitializer{
@Override
publicvoidonStartup(ServletContext servletContext)throws ServletException {
    WebApplicationContext context = getContext();
    servletContext.addListener(new ContextLoaderListener(context));
    ServletRegistration.Dynamic dispatcher =
servletContext.addServlet("DispatcherServlet", new DispatcherServlet(context));
    dispatcher.setLoadOnStartup(1);
    dispatcher.addMapping("/*");
```

```
    }
  private AnnotationConfigWebApplicationContext getContext(){
        AnnotationConfigWebApplicationContext context = new
  AnnotationConfigWebApplicationContext();
        context.setConfigLocation("eu.kielczewski.example.config");
  return context;
    }
  }
```

Several things happen here:

1. AnnotationConfigWebApplicationContext is created. It's WebApplicationContext implementation that looks for Spring configuration in classes annotated with @Configuration annotation. setConfigLocation() method gets hint in which package(s) to look for them.
2. ContextLoaderListener is added to ServletContext – the purpose of this is to 'glue' WebApplicationContext to the lifecycle of ServletContext.
3. DispatcherServlet is created and initialized with WebApplicationContext we have created, and it's mapped to "/*" URLs and set to eagerly load on application startup.

Configuration classes

The main AppConfig configuration class doesn't do anything but hits Spring on where to look for its components through @ComponentScan annotation.

```
@Configuration
@ComponentScan(basePackages = "eu.kielczewski.example")
public class AppConfig {
}
```

WebMvcConfig class enables Spring MVC with @EnableWebMvc annotation. It extends WebMvcConfigurerAdapter, which provides empty methods that can be overridden to customize default configuration of Spring MVC. We will stick to default configuration at this time, but it's advised for you to see what the possibilities are.

```
@EnableWebMvc
@Configuration
public classWebMvcConfigextendsWebMvcConfigurerAdapter {
}
```

Controller

Controllers are annotated with @Controller. It is found by Spring because of @ComponentScan annotation in AppConfig. The method will intercept GET request to "/" to which the response

will be sent. @ResponseBody indicates that whatever this method returns will be response body, and in this case it's just a "Hello world" String.

```
@Controller
public class IndexController {
@RequestMapping(value = "/", method = RequestMethod.GET)
@ResponseBody
   public String showIndex() {
return"Hello world";
   }
}
```

Example to show web page

index.jsp

```
<jsp:forward page="home.htm"/>
```

welcome.jsp

```
<h1><center> Welcome page</center></h1>
<br><br>
 ${sysDate }
```

RootConfig.java

```
package com.nt.cfgs;
import org.springframework.context.annotation.Configuration;
@Configuration
public class RootAppConfig {
}
```

WebMvcAppConfing.java

```
package com.nt.cfgs;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
@EnableWebMvc
@Configuration
@ComponentScan("com.nt.*")
public class WebMvcAppConfig {
    @Bean
    public ViewResolver createViewResolver(){
        InternalResourceViewResolver ivr=null;
        ivr=new InternalResourceViewResolver();
        ivr.setPrefix("/WEB-INF/pages/");
```

```
            ivr.setSuffix(".jsp");
            return ivr;
        }
}
```

ShowHomePageController.java

```
package com.nt.controller;
import java.util.Date;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ShowHomePageController {
        @RequestMapping("/home.htm")
        public String showHomePage(Map map){
            map.put("sysDate",new Date());

            return "welcome";
        }
}
```

WebAppInitializer.java

```
package com.nt.webinit;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;
import com.nt.cfgs.RootAppConfig;
import com.nt.cfgs.WebMvcAppConfig;
public class WebAppInitializer implements WebApplicationInitializer {
        @Override
        public void onStartup(ServletContext sc) throws ServletException {
            //create Root Application context Container
            AnnotationConfigWebApplicationContext rootCtx=new
AnnotationConfigWebApplicationContext();
            rootCtx.register(RootAppConfig.class);
            //create Web Application context container
            AnnotationConfigWebApplicationContext webCtx=new
AnnotationConfigWebApplicationContext();
            webCtx.register(WebMvcAppConfig.class);
            //create ContextLoaderListener
            ContextLoaderListener listener=new ContextLoaderListener(rootCtx);
            //add Listener to Servletcontxt object
            sc.addListener(listener);
            //Create DispatherServlet
            DispatcherServlet  servlet=new DispatcherServlet(webCtx);
            ServletRegistration.Dynamic  register=sc.addServlet("dispatcher",servlet);
```

```
            register.setLoadOnStartup(2);
            register.addMapping("*.htm");
        }
}
```

Deployment related initial activities:

a) Programmer deployes the application in the server -->ServletContext object will be created-->servlet container Spring ServletContainerInitializer class object by collecting it from jars added to WEB/lib folder--->onTrartup() method will be called on the above created object---> this method checks is their any class that implements WebAppInitializer- --> creates that class object and calls onStartup() method having ServletContext object in this method we place the logic of both RootContext, WEbAppContext containers having RootConfig, WebMvcAppConfig classes and these containers will be linked with ContextLoaderListener, DispatcherServlet class objects respectively and same listener, servlet will be registered with container dynamically.

Example Application that acts as AbstractController to process the request generated by hyperlink

index.jsp

```
<jsp:forward page="home.htm"/>
```

welcome.jsp

```
<h1><center> Welcome page</center></h1><br>
<a href="wish.htm">wish message</a>
```

result.jsp

```
<h1><centeR>Result Page</centeR></h1>
${resultMsg}
```

RootConfig.java

```
package com.nt.cfgs;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan("com.nt.service.*")
public class RootAppConfig {
}
```

WebMvcAppInitializer.java

```
package com.nt.cfgs;
import java.util.Properties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.HandlerMapping;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.handler.SimpleUrlHandlerMapping;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.mvc.ParameterizableViewController;
```

```
import org.springframework.web.servlet.view.InternalResourceViewResolver;
@EnableWebMvc
@Configuration
@ComponentScan("com.nt.*")
public class WebMvcAppConfig {

	@Bean
	public HandlerMapping  createHandlerMapping(){
		SimpleUrlHandlerMapping handlerMapping=new SimpleUrlHandlerMapping();
		Properties mappings=new Properties();
		mappings.put("/home.htm","pvc1");
		handlerMapping.setMappings(mappings);
		return handlerMapping;
	}
	@Bean(name="pvc1")
	public Controller  createController(){
		ParameterizableViewController pvc=new ParameterizableViewController();
		pvc.setViewName("welcome");
		return pvc;
	}

	@Bean
	public ViewResolver createViewResolver(){
		InternalResourceViewResolver ivr=null;

		ivr=new InternalResourceViewResolver();
		ivr.setPrefix("/WEB-INF/pages/");
		ivr.setSuffix(".jsp");
		return ivr;
	}
}
```

WebAppInitializer.java

```
package com.nt.webinit;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;
import com.nt.cfgs.RootAppConfig;
import com.nt.cfgs.WebMvcAppConfig;
public class WebAppInitializer implements WebApplicationInitializer {
	@Override
	public void onStartup(ServletContext sc) throws ServletException {
		//create Root Application context Container
		AnnotationConfigWebApplicationContext rootCtx=new
AnnotationConfigWebApplicationContext();
		rootCtx.register(RootAppConfig.class);
```

```
            //create Web Application context container
            AnnotationConfigWebApplicationContext webCtx=new
AnnotationConfigWebApplicationContext();
            webCtx.register(WebMvcAppConfig.class);
            //create ContextLoaderListener
            ContextLoaderListener listener=new ContextLoaderListener(rootCtx);
            //add Listener to Servletcontxt object
            sc.addListener(listener);
            //Create DispatherServlet
            DispatcherServlet  servlet=new DispatcherServlet(webCtx);
            ServletRegistration.Dynamic  register=sc.addServlet("dispatcher",servlet);
            register.setLoadOnStartup(2);
            register.addMapping("*.htm");
        }
}
```

WishController.java

```
package com.nt.controller;
import java.util.Map;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import com.nt.service.WishService;
@Controller
public class WishController {
        @Resource
        private WishService service;
        @RequestMapping("/wish.htm")
        public String processRequest(Map map){
                //use Service class
                String wishMsg=service.generateWishMsg();
                //add Model Data
                map.put("resultMsg",wishMsg);
                return "result";
        }
}
```

WishService.java

```
package com.nt.service;
public interface WishService {
        public String generateWishMsg();
}
```

WishServiceImpl.java

```
package com.nt.service;
import org.springframework.stereotype.Service;
@Service("ws")
public class WishServiceImpl implements WishService{
        @Override
        public String generateWishMsg() {
                return "Good Morning";
```

```
    }
}
```

Most people know that Spring's long supported Servlet 3 Java-based initializer classes. This facility builds on Servlet 3's javax.servlet.ServletContainerInitializer facility. Basically, you declare a class that implements org.springframework.web.WebApplicationInitializer and this class will be scanned by Spring on application startup and bootstrapped. This class has one responsibility: assemble the web application's moving parts, like you would in a web.xml, but in code. Here's an **example**:

```
public class MyWebAppInitializer implements WebApplicationInitializer {
   @Override
   public void onStartup(ServletContext container) {

      // Create the 'root' Spring application context
      AnnotationConfigWebApplicationContext rootContext = new
AnnotationConfigWebApplicationContext();
      rootContext.register(ServiceConfiguration.class);

      // Manage the lifecycle of the root application context
      container.addListener(new ContextLoaderListener(rootContext));

      // Create the dispatcher servlet's Spring application context
      AnnotationConfigWebApplicationContext dispatcherServlet = new
AnnotationConfigWebApplicationContext();
      dispatcherServlet.register(WebMvcConfiguration.class);

      // Register and map the dispatcher servlet
      ServletRegistration.Dynamic dispatcher = container.addServlet("dispatcher", new
DispatcherServlet(dispatcherServlet));
      dispatcher.setLoadOnStartup(1);
      dispatcher.addMapping("/");
   }
}
```

In the above example, WebConfiguration and ServiceConfiguration are Java configuratio classes. ServiceConfiguration presumably describes how the service tier - with its data source and transaction managers and thread pools and ORM mapping - are assembled. Th WebMvcConfiguration class describes how the web tier - supported by Spring MVC - is pl together. We use the ServletContext to build up the web application. But, a lot of this procedural. We will have basically the same arrangement each time, simply plugging different values as required.

The point of this post isn't to describe how Java configuration works, rather to introduce the WebApplicationInitializer implementation, called org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer

To use it, you simply extend it and satisfy fill in the blinks (in the form of template methods).

Here's an example:

```
import com.jl.crm.services.ServiceConfiguration;
import org.springframework.context.annotation.*;
import org.springframework.data.repository.support.DomainClassConverter;
import org.springframework.data.rest.webmvc.config.RepositoryRestMvcConfiguration;
import org.springframework.format.support.FormattingConversionService;
import org.springframework.hateoas.config.EnableHypermediaSupport;
import org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter;
import org.springframework.security.web.session.HttpSessionEventPublisher;
import org.springframework.web.filter.HiddenHttpMethodFilter;
import org.springframework.web.multipart.MultipartResolver;
import org.springframework.web.multipart.support.*;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.*;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

import javax.servlet.*;
import java.io.File;

public class CrmWebApplicationInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

   private int maxUploadSizeInMb = 5 * 1024 * 1024; // 5 MB

   @Override
   protected Class<?>[] getRootConfigClasses() {
      return new Class<?>[]{ServiceConfiguration.class};
   }

   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class<?>[]{RepositoryRestMvcConfiguration.class,
WebMvcConfiguration.class};
   }

   @Override
   protected String[] getServletMappings() {
      return new String[]{"/"};
   }

   @Override
   protected Filter[] getServletFilters() {
      return new Filter[]{new HiddenHttpMethodFilter(), new MultipartFilter(), new
OpenEntityManagerInViewFilter()};
   }
```

```
    @Override
    protected void registerDispatcherServlet(ServletContext servletContext) {
        super.registerDispatcherServlet(servletContext);

        servletContext.addListener(new HttpSessionEventPublisher());

    }

    @Override
    protected void customizeRegistration(ServletRegistration.Dynamic registration) {

        File uploadDirectory = ServiceConfiguration.CRM_STORAGE_UPLOADS_DIRECTORY;

        MultipartConfigElement multipartConfigElement =
            new MultipartConfigElement(uploadDirectory.getAbsolutePath(),
                maxUploadSizeInMb, maxUploadSizeInMb * 2, maxUploadSizeInMb / 2);

        registration.setMultipartConfig(multipartConfigElement);

    }
}
```

In typical Spring fashion, we have callbacks where Spring can't possibly devine our intents (it can't possibly know what the names of our configuration classes are, for example) but otherwise it's all taken care of for us. I had to override the default ServletRegistration.Dynamic instance so I could tailor where file uploads are stored by the Servlet 3 container, which I do in the customizeRegistration method. The rest of the code is fairly straightforward: Spring registers the Filters provided by the getServletFilters method, it registers the configuration classes returned from the getRootConfigurationClasses as part of a global ContextLoaderListener, and it registers the configuration classes returned from the getServletFilters as part of the DispatcherServlet. This reflects the typical layering in most Spring applications: global beans and services that are potentially to be shared among multiple DispatcherServlet implementations are stored in the ContextLoaderListener, and DispatcherServlet-local beans are configured there. My web configuration class is shown below. I show this mainly for completeness, but you can see I've registered and configured .jsp support, as well as multipart file upload support (taking advantage of the Servlet 3-powered StandardServletMultipartResolver).

Example Application on AbstractAnnotationConfigDispatcherServletInitializer

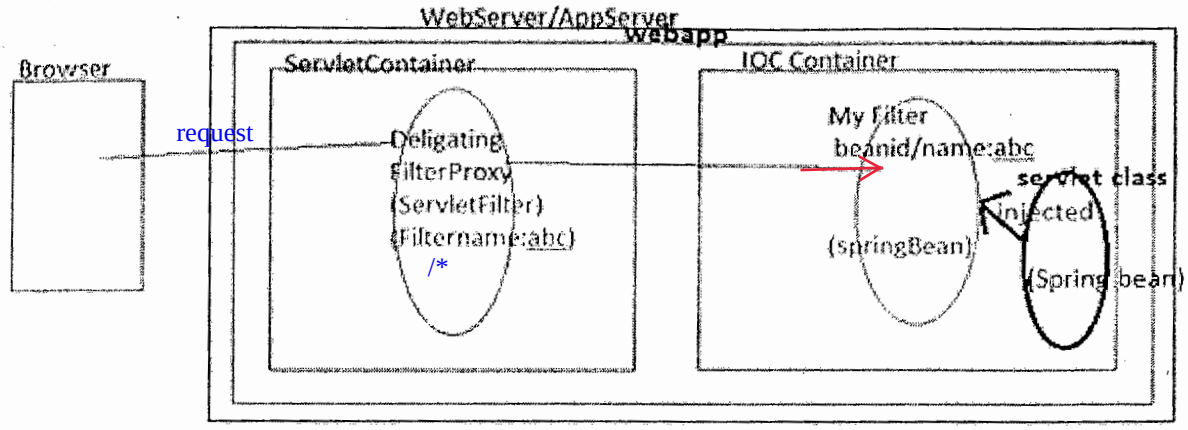
WebAppInitializer.java

```
package com.nt.webinit;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

import com.nt.cfgs.RootAppConfig;
import com.nt.cfgs.WebMvcAppConfig;

public class WebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{

	@Override
	protected Class<?>[] getRootConfigClasses() {
		return new Class[]{RootAppConfig.class};
	}

	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[]{WebMvcAppConfig.class};
	}

	@Override
	protected String[] getServletMappings() {
	  return new String[]{"*.htm"};

	}
}
```

Spring Security

To perform dependency injection on servletFilter component its class must be maintained by IOC container as spring bean having configuration in spring bean configuration file, Due to this it can't take request from client as filter is not configured as web component in web.xml file. To overcome this problem we configure , special proxy filter in web.xml file to take the request from client to pass the request to filter comp that is there in IOC container as spring bean, The special filter is "org.springframework.web.filter.DelegatingFilterProxy". But we need to match logical name (Given in <filter-name>) of DelegatingProxyFilter with bean name of filter comp managed by IOC container.

Diagram:-



Note:- My filter is spring bean here so we can inject other object to MyFilter…

Security (authentication + autherization)

- ⇨ Checking the identity of the user is called authentication (verifying username, password)
- ⇨ Checking the Access permission of the user to utilize the service is called authorization.

Provide authorization/access permission not based on username, provide them based on roles. =>Roles are like designations.

Checking the identity of user to use banking application is called authentication.

Checking access permission of a user based on his roles to the various services is called authorization.

While implementing security we need the following services.

a) Authentication provider/Authentication info provider

- ⇨ It is the place where we can maintain set of username and passwords that are required for authentication. We can take….

File system file (Text file, xml file etc….)
Database software
LDAP server (light weigth Directory Access protocol)
LDAP server is dedicaitedly given to maintain username and password.

- ⇨ The speciality of this server is once you forgot the password you can't get the password but you can reset the password. It is also to maintain username, password that required in single sign on operations (login once use them for multiple apps)

eg: Login to gmail and use in youtube.com, googleDrive.com, google+.com, gmail.com and etc..

b) Authentication Manager:

- ⇨ It is separate comp that is linked servletfilter to verify to given user, password wi username, password that are available in authentication provider. If valid allows t request going to the actually requested resource, If not valid shows 401, 403 error pages.

This has to ignore authentication when request is given some resource like home page and ha to enable authentication for other resources.

In non spring environment we can implement security in two ways.(Different Securi Models)

a) Programatic/Custom security model

Here we need to write code as Authentication manager that takes the request, render dialog boxes asking username, password and Talks to Authentication provider for verification and either allows the request to go to required resource or blocks the request by throwing 403 or 401 error page. Doing this work manually is very complex. So it is not recommended model.

b) **Server Managed declarative Security model:**

Here authentication manager work, authentication provider work will be taken care by server/container based on the configuration done in web.xml to reduce burden on the programmer but few limitations are there in this approach.

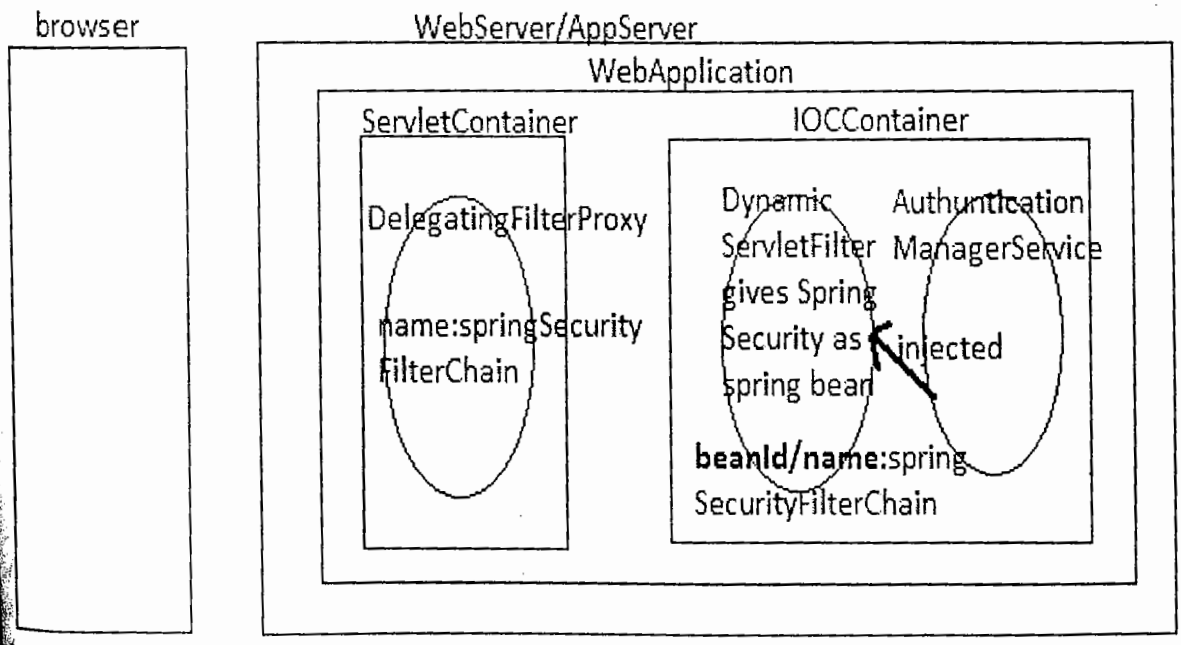
1) Web.xml entries are not portable accross the multiple servers.
2) Some web servers don't supports in this model.
3) Going to commercial Application server only for security model is cost effective.
4) Some server donot support LDAP server as authentication provider.

⇨ To overcome this problem use spring security whose earlier name is (Spring acegi security)(

1) It runs in all servers and works irrespective they support security or not.
2) Supports all models of authentication providers.
3) It is also declarative security model.
4) Can be applied on normal web application and spring mvc web application and etc...

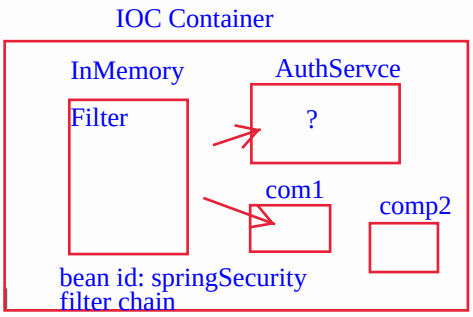
Note:

We can configure spring security definitions in springbean configuration file having security namespace, based on this configuration internally one ServletFilter will be generated as spring bean in IOC container having bean name,"springSecurityFilterChain" so to take request from client and to pass to request the above generated filter we need to configure "DelegatingFilterProxy" in web.xml with logical name "springSecurityFilterChain" .



SpringSecurityApp26

```
|----->java resources
        |----->webcontent
                |---->home.jsp
                |---->WEB-INF
                        |----->pages
                                |---->*.jsp
                        |----->applicationContext.xml
                |----->web.xml,
```



Jars:- MVC library+spring-security-config-3.0.5 RELEASE.jar,spring-security-core-3.0.5RELEASE.jar,

Spring-security-web-3.0.5RELEASE.jar(Gather from spring security separate installer and the ZIP name is:- spring-security-3.0.5RELEASE).

home.jsp

```
<p><a href="secure/Admin.jsp">Admin Page</a></p>
<p><a href="secure/Faculty.jsp">Faculty Page</a></p>
```

logout.jsp

```
You have successfully LoggedOut<br>
<a href="home.jsp">start again</a>
```

admin.jsp

```
<h1>Admin Page</h1>
Welcome Mr <%=request.getUserPrincipal().getName()%><br><br><br>
Wanna goto Faculty Page <a href="Faculty.jsp">click here</a>
<p><a href="../j_spring_security_logout">Logout</a>
```

faculty.jsp

```
<h1> Faculty Page </h1>
Welcome Mr <%=request.getUserPrincipal().getName()%><br><br><br>
Wanna goto Admin Page <a href="Admin.jsp">click here</a>
<p><a href="../j_spring_security_logout">Logout</a>
```

web.xm

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationcontext.xml</param-value>
</context-param>
<welcome-file-list>
<welcome-file>home.jsp</welcome-file>
</welcome-file-list>
<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>
</web-app>
```

applicationContext.xml

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
   xmlns:beans="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
              http://www.springframework.org/schema/security
            http://www.springframework.org/schema/security/spring-security-3.0.3.xsd">
<http use-expressions="true">
<intercept-url pattern="/home.jsp" access="permitAll" />
<intercept-url pattern="/secure/admin.jsp" access="hasRole('admin')" />
<intercept-url pattern="/secure/faculty.jsp" access="hasRole('faculty')" />
<form-login />
<logout />
<remember-me />
<session-management invalid-session-url="/logout.jsp">
   <concurrency-control max-sessions="3" error-if-maximum-exceeded="true" />
   </session-management>
</http>
<authentication-manager>
<authentication-provider>
<user-service>
<user name="raja" password="rani" authorities="admin,faculty" />
<user name="ravi" password="ravi" authorities="faculty" />
</user-service>
</authentication-provider>
</authentication-manager>
</beans:beans>
```

Spring Security Example Using Spring MVC

While working with 100% java code based spring MVC application we can write method in configuration class to get DataSource object from underlying servers JndiRegistery as shown below.

```
@Bean
public static DataSource create JndiDataSource(){
JndiDataSourceLookup dsLookup=null;
DataSource ds=null;
dsLookup=new JndiDataSourceLookup();
ds=dsLookup.getDataSource("java:/comp/env/DsJndi");
return ds;
}
```

SPRING ORM

Jdbc Technology allows the programmers to develop Database software dependent persistence logic by using SQL queries due to this jdbc code is not portable across the multiple Database softwares.
ORM allows to develop objects based Database software independent persistence logic due to this ORMMapping persistence logic is portable across the multiple Database softwares.

- The process of mapping java class with Database table java class properties with Database table cols and making java class objects representing Database table records having synchronization between them is called o-r mapping.

- Hibernate, ibatis, jdo, Toplink and etc... ORM softwares.

- Spring ORM is not direct ORM software. It is spring module that provides abstraction layer on multiple ORM software like Hibernate, iBatis, JDO, JPA etc.... and simplifies o-r mapping persistence logic development by providing template classes to avoide boilerplate code problem.

Spring ORM Advantages:

1) Common Transaction management : instead of performing each ORM software programmatic Transaction, management we can go for common spring style declarative Trans management.
2) Nested transction support: Most of the ORM s/w don't support Nested Transaction management directly. But spring supports Nested Transaction management . By using spring ORM we can add nested Transaction management support on ORM persistence logic.
3) Portability : gives portability to change one ORM software to another ORM software to develop persistence logic with minimum changes.
4) Avoids Boilerplatecode: with the support of multiple Template classes we can avoid boiler plate code (Repeated code)
5) CommonException handling: Instead of Handling each ORM specific checked and unchecked Exceptions spring ORM thows them as common "DataAccessException" hierarchy classes based exceptions and etc....

Spring Hibernate Integration (Spring with Hibernate):

- Spring with Hibernate application means developing Hibernate style object based OR mapping persistence logic by using HibernateTemplate class in the DAO class of spring application.
- To create HibernateTemplate object we need HB sessionFactory object. We can use FactoryBeans like "LocalSessionFactory" or "AnnotationSessinFactory" classes as beans to create sessionfactory object by supplying HB configure properties/HB configure file and mapping annotation based Domain class/HLO (Hibernate Language class) and also a Datasource object.This HibernateTemplate object will be injected to DAO class to perform persistence operation.
- HibernateTemplate class important methods are:
  - i) save(-), persist(-), update(-), load(-), get etc for single row operations.
  - ii) bulkUpdate(-) ---->to perform HQL based non-select operations.
  - iii) find(-) or findXxx(-) : To perform HQL/NativeSQL/criteris based bulk select operations.

Database Table:

SPEmployee

|---->eno number pk
|---->ename varchar
|----->salary float

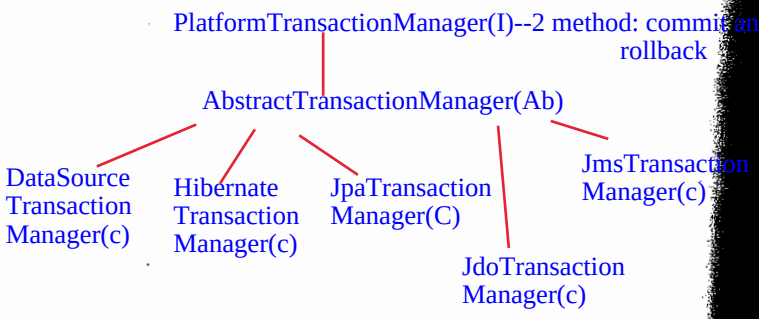
⇨ while importing certain framework/technology main jar file if you want to exclude certain unnecessary dependent jar files using maventhen you can write <exclusions>, <exclusion> tags in pom.xml file under dependency tag as shown below.

```
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.1.0.Final</version>
<exclusions>
<exclusion>
<groupId>org.jboss</groupId>
<artifactId>jandex</artifactId>
</exclusion>
</exclusions>
</dependency>
```

// delete below example and add new example from sir blog

- ORMProj1(Spring 4withHibernate3)
  - src
    - com.nt.cfgs
      - aop-beans.xml
      - applicationContext.xml
      - dispatcher-servlet.xml
      - persistence-beans.xml
      - service-beans.xml
    - com.nt.controller
      - EmployeeController.java
    - com.nt.dao
      - EmployeeDAO.java
      - EmployeeDAOImpl.java
    - com.nt.domain
      - EmployeeHLO.java
    - com.nt.dto
      - EmployeeDTO.java
    - com.nt.service
      - EmployeeService.java
      - EmployeeServiceImpl.java
    - test
      - ClientApp.java
  - JRE System Library [JavaSE-1.7]

when we are using Hibernate Template class save(),update(); and delete methods with spring 4.x and hibernate 4.x/5.x then we get the following exception InvalidDataAccessApiusageException: write oparations are not allowed in read-only mode. to resolve this problem we can configure hibernateTransactionManager
obj explicitly and we can use @Transactional annotation with readOnly=false.



Jar IOC: jars + Hibernate 3.6.5 jars+ AOP jar+ ojdbc 14.jar+Apache DPCP jar+Spring-ORM-4+SpjDBC jar+aspects.jar+aopaliance+cglib
(Hibernate 3.6.5 jars= Hibernate 3.6.5 Home/Hibernate 3.0,lib folder all jar files,lib –JPA folder 1 jar file).

aop-beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">


<bean id="txMgmr"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
<property name="sessionFactory" ref="sesfact"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="txMgmr">
<tx:attributes>
```

```
<tx:method name="register"  propagation="REQUIRED"
                                                read-only="false"/>
</tx:attributes>
</tx:advice>

<aop:config>
<aop:pointcut id="ptc1"
                    expression="execution(* com.nt.service.EmployeeService.*(..))"/>
<aop:advisor advice-ref="txAdvice" pointcut-ref="ptc1"/>
</aop:config>

</beans>
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<import resource="service-beans.xml"/>
<import resource="persistence-beans.xml"/>
<import resource="aop-beans.xml"/>

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

</beans>
```

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<bean
class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>

<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
<property name="basename" value="com/nt/commons/errors"/>
</bean>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix"  value="/WEB-INF/pages/"/>
```

```
<property name="suffix"  value=".jsp"/>
</bean>

<context:component-scan base-package="com.nt.controller,com.nt.validator"/>

</beans>
```

persistence-beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<bean id="dbcpds"  class="org.apache.commons.dbcp.BasicDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="scott"/>
<property name="password" value="tiger"/>
</bean>
<bean id="sesfact"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
<property name="dataSource" ref="dbcpds"/>
<property name="annotatedClasses">
<list>
<value>com.nt.domain.EmployeeHLO</value>
</list>
</property>
<property name="hibernateProperties">
<props>
<prop key="show_sql">true</prop>
<prop key="dialect">org.hibernate.dialect.Oracle10gDialect</prop>
</props>
</property>
</bean>

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

<context:component-scan base-package="com.nt.dao"/>

</beans>
```

service-beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.nt.service"/>

</beans>
```

EmployeeController.java

```
package com.nt.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.nt.dto.EmployeeDTO;
import com.nt.service.EmployeeService;

@Component("controller")
publicclass EmployeeController {

	@Autowired
	private EmployeeService  service;


	public String registerEmployee(String name,float salary){
		//create DTO class object
		EmployeeDTO empDTO=new EmployeeDTO();
		empDTO.setEname(name);
		empDTO.setEsalary(salary);
		//use service class
		String result=service.register(empDTO);
		return result;
	}

	public  List<EmployeeDTO> listEmployees(){
		//use service class
		List<EmployeeDTO> listEmpDTO=service.listEmps();
		return listEmpDTO;
	}

}
```

EmployeeDAO.java

```
package com.nt.dao;

import java.util.List;

import com.nt.domain.EmployeeHLO;

public interface EmployeeDAO {
	public int  insert(EmployeeHLO empHLO );
	public List<EmployeeHLO> getAllEmployees();

}
```

EmployeeDAOImpl.java

```
package com.nt.dao;

import java.sql.SQLException;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.SQLQuery;
import org.hibernate.Session;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate3.HibernateCallback;
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.stereotype.Repository;

import com.nt.domain.EmployeeHLO;

@Repository
publicclass EmployeeDAOImpl implements EmployeeDAO{

	@Autowired
	private HibernateTemplate ht;

	@Override
	publicint insert(EmployeeHLO empHLO) {
		int id=(Integer)ht.save(empHLO);
		return id;
	}

	@Override
	public List<EmployeeHLO> getAllEmployees() {

	List<EmployeeHLO>list=ht.execute(new HibernateCallback  <List<EmployeeHLO>>(){

			@Override
			public List<EmployeeHLO> doInHibernate(Session ses)
				throws HibernateException, SQLException {
			SQLQuery query=ses.createSQLQuery("select * from
SpEmployee");
			query.addEntity(EmployeeHLO.class);
			List<EmployeeHLO> list=query.list();
			return list;
			}
		});
```

```
        return list;
        }
}
```

EmployeeHLO.java

```
package com.nt.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import org.hibernate.annotations.GenericGenerator;

@Table(name="SpEmployee")
@Entity
@NamedQuery(name="GET_EMPS_BY_SAL_RANGE",
                    query="from EmployeeHLO where esalary>=:min")
publicclass EmployeeHLO {
        @Id
        @GenericGenerator(strategy="increment", name="g1")
        @GeneratedValue(generator="g1")
        privateint eno;
        private String ename;
        privatefloat esalary;
        publicint getEno() {
                return eno;
        }
        publicvoid setEno(int eno) {
                this.eno = eno;
        }
        public String getEname() {
                return ename;
        }
        publicvoid setEname(String ename) {
                this.ename = ename;
        }
        publicfloat getEsalary() {
                return esalary;
        }
        publicvoid setEsalary(float esalary) {
                this.esalary = esalary;
        }

}
```

EmployeeDTO.java

```
package com.nt.dto;

import java.io.Serializable;

publicclass EmployeeDTO implements Serializable {
        privateint eno;
        private String ename;
        privatefloat esalary;
        publicint getEno() {
```

```
		return eno;
	}
	publicvoid setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	publicvoid setEname(String ename) {
		this.ename = ename;
	}
	publicfloat getEsalary() {
		return esalary;
	}
	publicvoid setEsalary(float esalary) {
		this.esalary = esalary;
	}
	@Override
	public String toString() {
		return "EmployeeDTO [eno=" + eno + ", ename=" + ename + ", esalary="
				+ esalary + "]";
	}
}
```

EmployeeService.java

```
package com.nt.service;

import java.util.List;

import com.nt.dto.EmployeeDTO;

publicinterface EmployeeService {

	public String register(EmployeeDTO empDTO);

	public List<EmployeeDTO> listEmps();

}
```

EmployeeServiceImpl.java

```
package com.nt.service;

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

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

import com.nt.dao.EmployeeDAO;
import com.nt.domain.EmployeeHLO;
import com.nt.dto.EmployeeDTO;

@Service
publicclass EmployeeServiceImpl implements EmployeeService {

	@Autowired
```

```
        private EmployeeDAO dao;

        @Override
        public String register(EmployeeDTO empDTO) {
                //Covert DTO object to HLO object/Domain object
                EmployeeHLO empHLO=new EmployeeHLO();
                empHLO.setEname(empDTO.getEname());
                empHLO.setEsalary(empDTO.getEsalary());
                //use DAO
                int id=dao.insert(empHLO);
                        return "Emp  Registred with "+id;
        }

        @Override
        public List<EmployeeDTO> listEmps() {
                //use service DAO class
                List<EmployeeHLO> listEmpHLO=dao.getAllEmployees();
                //convert ListHLO to ListDTO
                List<EmployeeDTO> listEmpDTO=new ArrayList<EmployeeDTO>();
                for(EmployeeHLO hlo:listEmpHLO){
                        EmployeeDTO dto=new EmployeeDTO();
                        dto.setEno(hlo.getEno());
                        dto.setEname(hlo.getEname());
                        dto.setEsalary(hlo.getEsalary());
                        listEmpDTO.add(dto);
                }

                return listEmpDTO;
        }
}
```

ClientApp.java

```
package test;

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

import com.nt.controller.EmployeeController;

publicclass ClientApp {

        publicstaticvoid main(String[] args) {
        //create IOC container
                ApplicationContext ctx=new
FileSystemXmlApplicationContext("src/com/nt/cfgs/applicationContext.xml");
        //get Bean (Controller)
                EmployeeController
controller=ctx.getBean("controller",EmployeeController.class);
        //call b.method
                //System.out.println(controller.registerEmployee("raja",9000));

                 System.out.println("Emp Details are"+controller.listEmployees());

        }
}
```

Plain hibernate application

1) create configuration object and call configure() method. |
2) Create service registry object by using serviceRegistryBuilder | //Common logics
3) Create Sessionfactory object |
4) Open session with Database software |
5) Begin Transaction. |
6) Perform persistence operation ----------->//Application specific logics
7) Commit or Rollback Transaction management
8) Close session, sessionfactory object

NOTE:

In plain HB application we need to write both common and application specific logics.

Spring with Hibernate application (Using spring ORM)

1) Inject HibernateTemplate object
2) Perform persistence operation.

⇨ **Executing HQL queries with named parameters:**

List<EmployeeHLO> list=(List<EmployeeHLO>)ht. findByNamedParam ("from EmployeeHLO where salary>=:min and salary<=:max", new String [] {"min","max"}, new Object[] {9000, 10000});

⇨ **Executing HQL query with positional parameters:**

List<EmployeeHLO> list=(List<EmployeeHLO>)ht.find("from EmpolyeeHLO where salary>=? and salary<=?", 9000.0f, 10000.0f);

While working with Hibernate we place Nativesql/HQL query in mapping file or domain class as named queries to make them visible in multiple session objects of one or more DAO class.

⇨ **Executing named HQLQuery with positional parameters:**

In EmployeeHLO.java (Domain Class)

@NamedQuery(name=""GET_EMPS_BY_SAL_RANGE", QUERY="from EmployeeHLO where salary>=? and salary<=?")

In DAO Class Method

List<EmployeeHLO>List=(ListEmployeeHLO)ht.findByNamedQuery("GET_EMPS_BY_SAL_RANGE", 9000.0F,10000.0F);

⇨ **Executing Named HQLQuery with Named parameters:**

In Domain class (EmployeeHLO.java)

@NamedQuery(name="GET_EMPS_BY_SAL_RANG", Query="from EmployeeHLO where salary>=:min")

In DAO class

List <EmployeeHLO>list=(List<EmployeeHLO>)ht.findNamedQueryAndNmaedParam("GETEMPS_BY_SAL_RNG", new String[]{"min"} new Object[] {9000.0f}));

If HibernateTemplate class doesn't provide envinorment to use certain feature of plain Hibernate, then take the support of Hibernate Callback interface, that gives doInHibernate(Session session) method. Using this object we can finish that operation.

For example

We can't use native Sql query using HibernateTemplate, to make it possible we can use this Hibernate Callback Interface support.

eg:

```
List <EmployeeHLO> list=ht.execute(new HibernateCallback<List<EmployeeHLO>>())
{
@Override
public List<EmployeeHLO> doInHibernate(Session ses) throws HibernateException,
SQLException
{
//Plain Hb code
SQLQuery query=ses.createSQLQuery("Select * from spemployee");
query.addEntity(EmployeeHLO.class);
List<EmployeeHLO>list=query.list();
return list;
}
}
```

NOTE:-

To perform TransactionManagement on Spring ORM based Hibernate persistence logics we should use HibernateTransactionManager which can be configured by injecting HibernateSessionFactoryObject.

Executing Named HQL queries with NamedParameters.(for more visibility of query)

Place NamedQuery in the domain class

```
@NamedQuery (name="AUTH_QUERY",query="SELECT COUNT(*) FROM UserHLO WHERE
UNAME=:un and pwd=:pass")
Public class UserHLO{
// logic
}
```

In DAO class (write following code)

```
List<?> list=ht.findNamedQueryAndNmaedParam("AUTH_QUERY", new String[]{"un","pass"}
new String[] {bo.getUname(),bo.getPwd()});
```

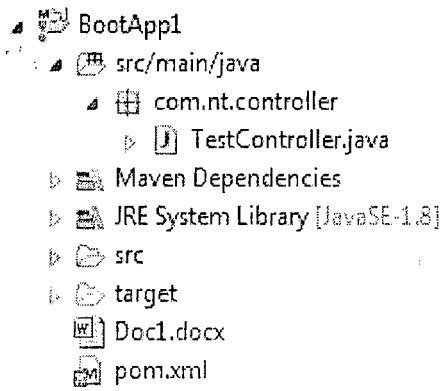
Executing HQL non select query w ith positional parameters:

```
Int cnt=ht.bulkUpdate("delete from UserHLO where unmae=?", "raja");
    System.out.println("records effected:"+ cnt);
        return cnt
```

==========00000000==========

SpringBoot

eg:



TestController.java

```
package com.nt.controller;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@EnableAutoConfiguration
publicclass TestController {

        @RequestMapping("/")
        public String process(){
                return "welcome to Spring Boot";
        }

        publicstaticvoid main(String[] args) throws Exception {
      SpringApplication.run(com.nt.controller.TestController.class, args);
   }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
_____xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>NIT</groupId>
        <artifactId>BootApp1</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.0.0.BUILD-SNAPSHOT</version>
        </parent>

        <dependencies>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
    <!-- Additional lines to be added here... -->

    <!-- (you don't need this if you are using a .RELEASE version) -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

SpringBatch

eg:

- SpringBatchApp1
  - src/main/java
    - com.nt.batch
      - CustomItemProcessor.java
      - CustomItemReader.java
      - CustomItemWriter.java
    - com.nt.cfgs
      - applicationContext.xml
      - simple-job.xml
    - NIT.SpringBatchApp1
      - App.java
  - src/test/java
  - Maven Dependencies
  - JRE System Library [jre1.8.0_20]
  - src
  - target
  - pom.xml

CustomItemProcessor.java

```
package com.nt.batch;
import org.springframework.batch.item.ItemProcessor;

publicclass CustomItemProcessor implements ItemProcessor<String, String> {

	@Override
	public String process(String bookNameWithoutAuthor) throws Exception {
		System.out.println("ItemProcessor:process(-)");
		String bookNameWithAuthor = "Book Name - "+bookNameWithoutAuthor+" |
Author Name - ";

		if("Core Java".equalsIgnoreCase(bookNameWithoutAuthor)){
			bookNameWithAuthor += "Sriniwas";
		}elseif("Design Patterns".equalsIgnoreCase(bookNameWithoutAuthor)){
			bookNameWithAuthor += "raja ";
		}elseif("Advance Java".equalsIgnoreCase(bookNameWithoutAuthor)){
			bookNameWithAuthor += "ravi ";
		}elseif("Spring FrameWork".equalsIgnoreCase(bookNameWithoutAuthor)){
			bookNameWithAuthor += "karan";
		}elseif("Hibernate Framework".equalsIgnoreCase(bookNameWithoutAuthor)){
			bookNameWithAuthor += "rani";
		}
		return bookNameWithAuthor;
	}

}
```

CustomItemReader.java

```
package com.nt.batch;

import java.util.List;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ParseException;
import org.springframework.batch.item.UnexpectedInputException;

publicclass CustomItemReader implements ItemReader<String>{
	private List<String> bookNameList;
	privateint bookCount = 0;

	@Override
	public String read() throws Exception, UnexpectedInputException,
			ParseException {
		System.out.println("ItemReader::read()");
		if(bookCount < bookNameList.size()){
			return bookNameList.get(bookCount++);
		}else{
			returnnull;
		}
	}
	/*public List<String> getUserNameList() {
		return bookNameList;
	}*/
	publicvoid setBookNameList(List<String> bookNameList) {
		this.bookNameList = bookNameList;
	}

}
```

CustomItemWriter.java

```
package com.nt.batch;

import java.util.List;
import org.springframework.batch.item.ItemWriter;

publicclass CustomItemWriter implements ItemWriter<String> {

	@Override
	publicvoid write(List<? extends String> bookNameWithAuthor) throws Exception {
		System.out.println("ItemWriter: write(-)");
		System.out.println(bookNameWithAuthor);
	}
}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	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/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">


	<bean id="transactionManager"
	class="org.springframework.batch.support.transaction.ResourcelessTransactionManager"
/>

<bean id="jobRepository"

class="org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean">
<property name="transactionManager" ref="transactionManager"/>
</bean>

<bean id="jobLauncher"
class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
<property name="jobRepository" ref="jobRepository"/>
</bean>

<bean id="simpleJob"
   class="org.springframework.batch.core.job.SimpleJob" abstract="true">
<property name="jobRepository" ref="jobRepository" />
</bean>

</beans>
```

simple-job.xml

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

```
        xmlns:batch="http://www.springframework.org/schema/batch"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        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/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
http://www.springframework.org/schema/batch
http://www.springframework.org/schema/batch/spring-batch-3.0.xsd">

        <import resource="applicationContext.xml"/>

        <bean id="customReader" class="com.nt.batch.CustomItemReader">
<property name="bookNameList">
<list>
<value>Core Java</value>
<value>Design Patterns</value>
<value>Advance Java</value>
<value>Spring FrameWork</value>
<value>Hibernate Framework</value>
</list>
</property>
</bean>

<bean id="customProcessor" class="com.nt.batch.CustomItemProcessor" />

<bean id="customWriter" class="com.nt.batch.CustomItemWriter" />

<batch:job id="mySimpleJob" job-repository="jobRepository" parent="simpleJob">
        <batch:step id="step1">
                <batch:tasklet transaction-manager="transactionManager">
                        <batch:chunk reader="customReader" processor="customProcessor"
                                        writer="customWriter" commit-interval="2"/>
                </batch:tasklet>
        </batch:step>
</batch:job>
</beans>
```

App.java

```
package NIT.SpringBatchApp1;

/**
 * Hello world!
 *
 */
publicclass App
{
publicstaticvoid main( String[] args )
   {
      System.out.println( "Hello World!" );
   }
}
```

pom.xml

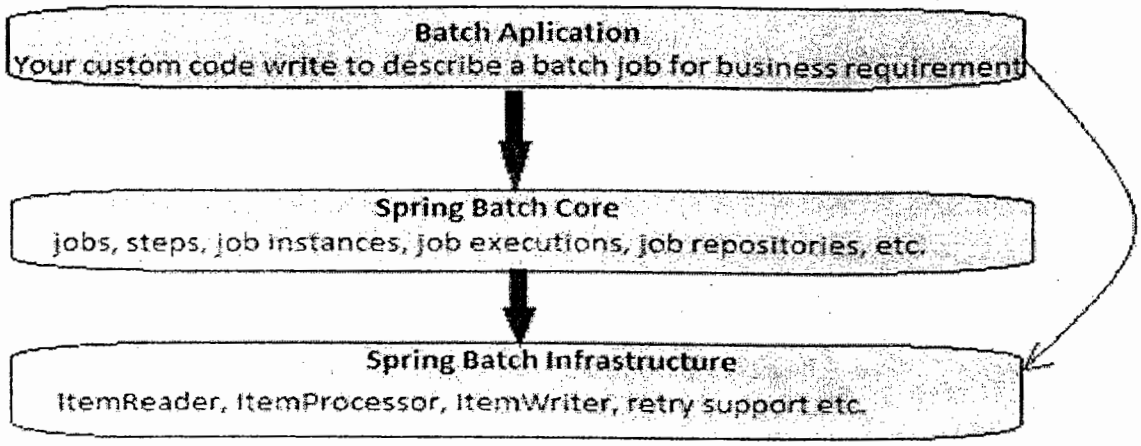
```
<project
xmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchem
a-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>NIT</groupId>
<artifactId>SpringBatchApp1</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>SpringBatchApp1</name>
<url>http://maven.apache.org</url>

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>4.2.2.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>4.2.2.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context-support</artifactId>
<version>4.2.2.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.batch/spring-batch-core -->
<dependency>
<groupId>org.springframework.batch</groupId>
<artifactId>spring-batch-core</artifactId>
<version>3.0.7.RELEASE</version>
</dependency>

</dependencies>
</project>
```

Spring Batch Processing is defined as the Processing of Data without Interaction or interruption

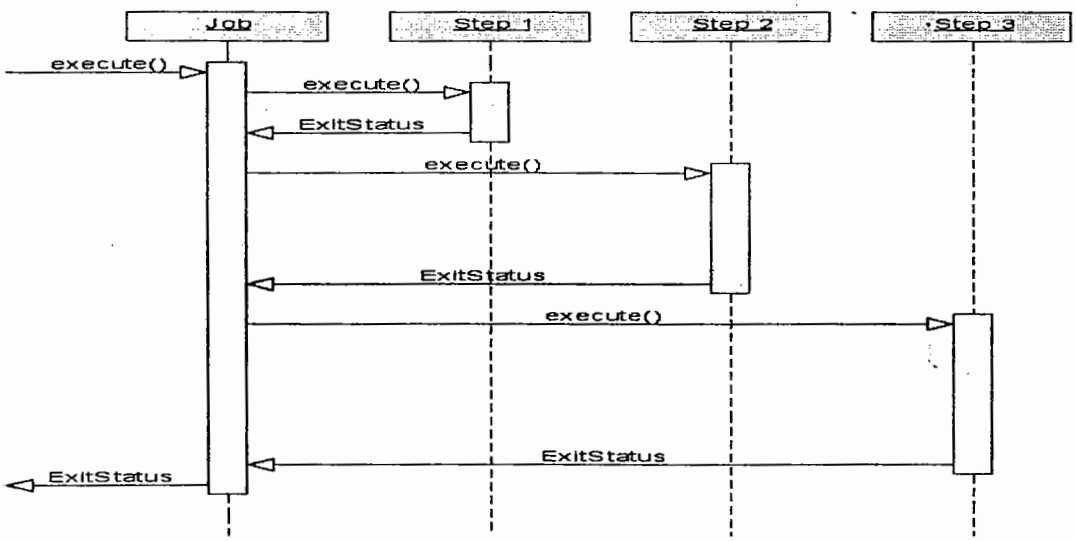
Detailed Information:-

1. Spring Batch is an open-source framework for batch processing. Batch Processing in simple terms, refers to running bulk operations that could run for hours on end without needing human intervention. Consider Enterprise level operations that involve say, reading from or writing into or updating millions of database records. Spring Batch provides the framework to have such jobs running with minimum human involvement.
2. It is light-weight, comprehensive, favors POJO-based development approach and comes with all the features that spring offers. Besides, it also exposes a number of classes and APIs that could be exploited say for transaction management, for reading and writing data.
3. Spring Batch provides reusable functions that are essential in processing large volumes of records, including logging/tracing, transaction management, job processing statistics, job restart, skip, resource management, logging, tracing, conversion of data, interfaces, etc. By using these diverse techniques, the framework takes care of the performance and the scalability while processing the records.
4. Spring Batch also provides more advanced technical services and features that will enable extremely high-volume and high performance batch jobs through optimization and partitioning techniques.
5. Spring Batch Application executes a series of jobs (iterative or in parallel), where input data is read, processed and written without any interaction. We are going to see how Spring Batch can help us with this purpose.
6. Normally a batch application can be divided in three main parts:
   - Reading the data (from a database, file system, etc.)
   - Processing the data (filtering, grouping, calculating, validating...)
   - Writing the data (to a database, reporting, distributing...)
7. Spring Batch contains features and abstractions for automating these basic steps and allowing the application programmers to configure them, repeat them, retry them, stop them, executing them as a single element or grouped (transaction management), etc.
8. It also contains classes and interfaces for the main data formats, industry standards and providers like XML, CSV, SQL, Mongo DB, etc.
9. **Examples :-**
   Month-end calculations notices or correspondence , periodic application of complex business rules processed repetitively across very large data sets (e.g. insurance benefit determination or rate adjustments)

HIGH LEVEL ARCHITECTURE



In above figure, the top of the hierarchy is the batch application itself. This is whatever batch processing application you want to write. It depends on the Spring Batch core module, which primarily provides a runtime environment for your batch jobs. Both the batch app and the core module in turn depend upon an infrastructure module that provides classes useful for both building and running batch apps.



As we see in above figure a hypothetical three-step job, though obviously a job can have arbitrarily many steps. The steps are typically sequential, though as of Spring Batch 2.0 it's possible to define conditional flows (e.g., execute step 2 if step 1 succeeds; otherwise execute step 3). Within any given step, the basic process is as follows: read a bunch of "items" (e.g., database rows, XML elements, and lines in a flat file— whatever), process them, and write them out somewhere to make it convenient for subsequent steps to work with the result.

Example- ((Spring Batch ItemReader and ItemWriter Example))

Intro:-

We will discuss about the three most important interfaces of spring batch and an overview of Spring Batch item reader and writer with a sample application. One of the important goals of a batch processing framework is to read large amounts of data, perform some business processing/transformation and write out the result. **Spring Batch Framework** supports this

bulk reading, processing and writing using three key interfaces: **ItemReader, ItemProcessor and ItemWriter**.

1. ItemReader is the means for providing data from many different types of input. ItemReader interface is the means for reading bulk data in a bulk processing system. There are many different implementations of ItemReader interface. All implementations are expected to be stateful and will be called multiple times for each batch, with each call to read() returning a different value and finally returning null when all input data is exhausted. Below are few frequently used implementations of ItemReader.

ItemReader Implementation	Description
FlatFileItemReader	Reads lines of data from input file. Typically read line describe records with fields of data defined by fixed positions in the file or delimited by some special character (e.g. Comma (,), Pipe (|) etc).
JdbcCursorItemReader	Opens a JDBC cursor and continually retrieves the next row in the ResultSet.
StoredProcedureItemReader	Executes a stored procedure and then reads the returned cursor and continually retrieves the next row in the ResultSet.

All the above implementations override the read() method from the ItemReader interface. The read method defines the most essential contract of the ItemReader. It returns one item or null if no more items are left. An item might represent a line in a file, a row in a database and so on.

2. ItemWriter is similar in functionality to an ItemReader, but with inverse operations. ItemWriter is an interface for generic output operations. Implementation class will be responsible for serializing objects as necessary. Resources still need to be located, opened and closed but they differ in that an ItemWriter writes out, rather than reading in. For databases these may be inserts or updates.

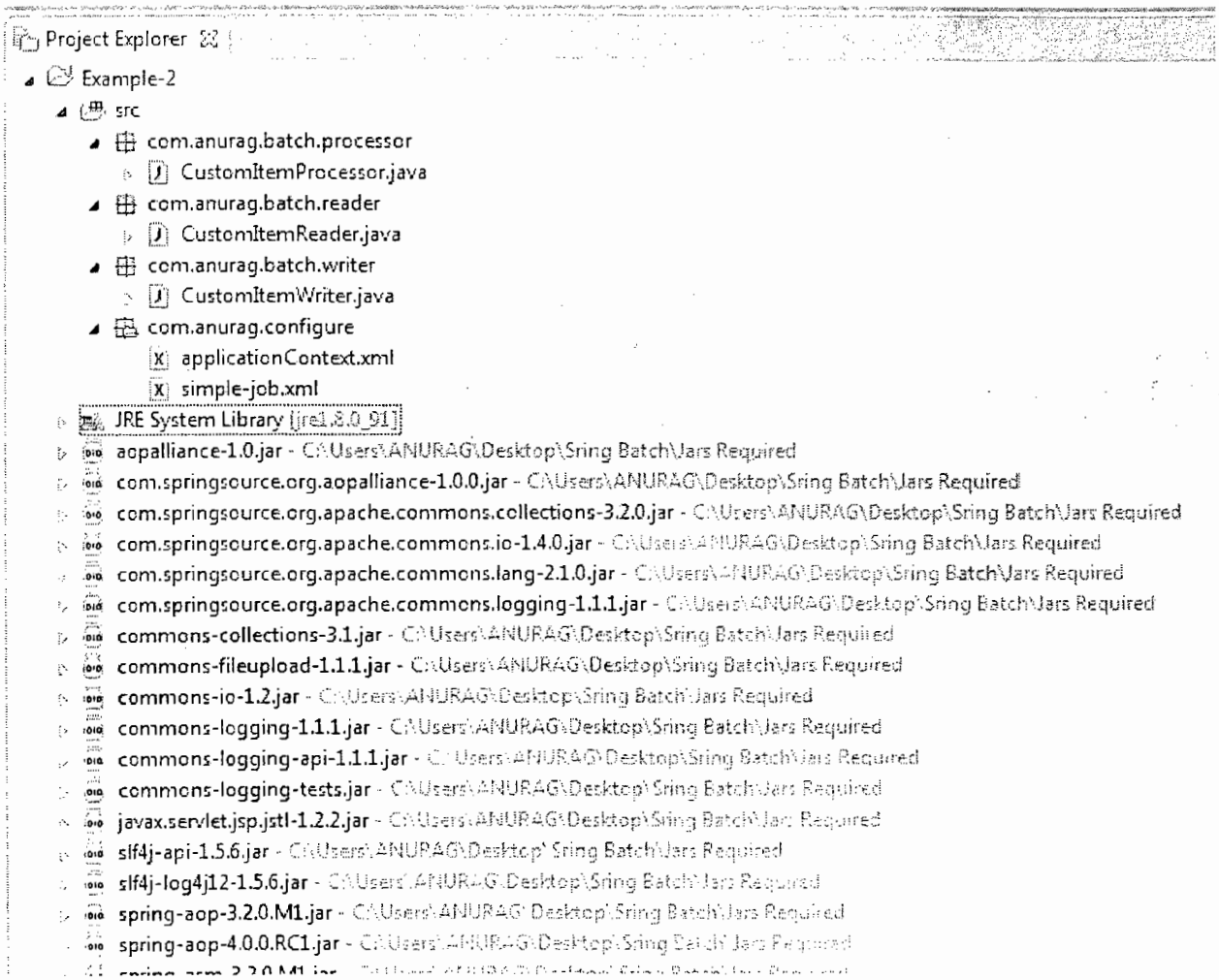
The write method defines the most essential contract of the ItemWriter. It will attempt to write out the list of items passed in as long as it is open. As it is expected that items will be 'batched' together into a chunk and then output, the interface accepts a list of items, rather than an item by itself. Once the items are written out, any flushing that may be necessary can be performed before returning from the write method.

ItemWriter Implementation	Description
FlatFileItemWriter	Writes data to a file or stream. Uses buffered writer to improve performance.
StaxEventItemWriter	An implementation of ItemWriter which uses StAX and Marshaller for serializing object to XML.

3. <u>ItemProcessor</u>- The ItemReader and ItemWriter interfaces are both very useful for their specific tasks, but what if you want to insert business logic before writing? An ItemProcessor is very simple interface for item transformation. Given one object, transform it and return another. Any business/transformation logic can be plugged into this component. Assume an ItemReader provides a class of type User, and it needs to be converted to type Employee before being written out. An ItemProcessor can be written that performs the conversion. Another typical use for an item processor is to filter out records before they are passed to the ItemWriter. Filtering simply indicates that a record should not be written.

Project Directory Structure-

Review the final project structure, a standard java project.



Note: - Add All the Jars of Jars Required folder

Create Custom Item Reader-

Below is our custom item reader. Each time it is called, it returns the next element from the list and returns null if the list is exhausted.

```
CustomItemProcessor.java  *CustomItemReader.java  CustomItemWriter.java  applicationContext.xml  simple-job.xml
 3 import java.util.List;
 8
10 * @author Anurag Kumar
14 public class CustomItemReader implements ItemReader<String>{
15
16     private List<String> bookNameList;
17     private int bookCount = 0;
18     @Override
19     public String read() throws Exception, UnexpectedInputException,
20             ParseException {
21         if(bookCount < bookNameList.size()){
22             return bookNameList.get(bookCount++);
23         }else{
24             return null;
25         }
26     }
27     public List<String> getUserNameList() {
28         return bookNameList;
29     }
30     public void setBookNameList(List<String> bookNameList) {
31         this.bookNameList = bookNameList;
32     }
33
34 }
```

Create Custom Item Processor-

CustomItemProcessor is simple custom ***ItemProcessor*** which transforms every element returned by the ItemReader. Here book name return with respective author name.

```
CustomItemProcessor.java  *CustomItemReader.java  CustomItemWriter.java  applicationContext.xml  simple-job.xml
 3 import org.springframework.batch.item.ItemProcessor;
 4
 6 * @author Anurag Kumar
10 public class CustomItemProcessor implements ItemProcessor<String, String> {
11
12     @Override
13     public String process(String bookNameWithoutAuthor) throws Exception {
14         String bookNameWithAuthor = "Book Name - "+bookNameWithoutAuthor+" | Author Name - ";
15         if("Core Java".equalsIgnoreCase(bookNameWithoutAuthor)){
16             bookNameWithAuthor += "Sriniwas";
17         }else if("Design Patterns".equalsIgnoreCase(bookNameWithoutAuthor)){
18             bookNameWithAuthor += "Natraz Design Patterns";
19         }else if("Advance Java".equalsIgnoreCase(bookNameWithoutAuthor)){
20             bookNameWithAuthor += "Natraz Adv. Java";
21         }else if("Spring FrameWork".equalsIgnoreCase(bookNameWithoutAuthor)){
22             bookNameWithAuthor += "Natraz Spring";
23         }else if("Hibernate Framework".equalsIgnoreCase(bookNameWithoutAuthor)){
24             bookNameWithAuthor += "Natraz Hibernate";
25         }
26         return bookNameWithAuthor;
27     }
28
29 }
30
```

Create Custom Item Writer-

CustomItemWriter is custom ItemWriter which outputs the transformed item(s) returned by our **CustomItemProcessor**.

```
CustomItemProcessor.java   CustomItemReader.java   CustomItemWriter.java   applicationContext.xml   sample-job.xml
 1 package com.anurag.batch.writer;
 2
 3* import java.util.List;
 6
 8* * @author Anurag Kumar
12 public class CustomItemWriter implements ItemWriter<String> {
13
14     @Override
15     public void write(List<? extends String> bookNameWithAuthor) throws Exception {
16         System.out.println(bookNameWithAuthor);
17     }
18
19 }
20
```

Create Application Context XML-

Below is the ***applicationContext.xml*** which is required to create JobRepository, JobLauncher and TransactionManager.

JobRepository

Repository is responsible for persistence of batch meta-data information. SimpleJobRepository is an implementation ofJobRepository that stores JobInstances, JobExecutions, and StepExecutions information using the DAOs injected via constructor arguments. Spring Batch supports two implementation of these DAOs: Map based (in-memory) and Jdbc based. In real enterprise application the Jdbc variants are preffered but we will use simpler in-memory alternatives (MapJobInstanceDao,MapJobExecutionDao, MapStepExecutionDao, MapExecutionContextDao) in this example.

JobLauncher

As name suggests it is responsible for launching batch job. We are using SimpleJobLauncher implementation which requires only one dependency, a JobRepository. JobRepository is used to obtain a valid JobExecution. Repository must be used because the provided Job could be a restart of an existing JobInstance, and only the Repository can reliably recreate it.

TransactionManager

As this example won't be dealing with transactional data, we are using ResourcelessTransactionManager which is mainly used for testing purpose.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    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/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <bean id="transactionManager"
    class="org.springframework.batch.support.transaction.ResourcelessTransactionManager"/>

    <bean id="jobLauncher" class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
        <property name="jobRepository" ref="jobRepository"/>
    </bean>

    <bean id="jobRepository"
    class="org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean">
        <property name="transactionManager" ref="transactionManager"/>
    </bean>

    <bean id="simpleJob"
    class="org.springframework.batch.core.job.SimpleJob" abstract="true">
        <property name="jobRepository" ref="jobRepository" />
    </bean>

</beans>
```

Create Job configuration XML-

It's time to wire the above 3 components together into a job which will perform reading, processing and transformation work for us.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:batch="http://www.springframework.org/schema/batch"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    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/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
http://www.springframework.org/schema/batch
http://www.springframework.org/schema/batch/spring-batch-2.0.xsd">

    <import resource="applicationContext.xml"/>

    <bean id="customReader" class="com.anurag.batch.reader.CustomItemReader" >
        <property name="bookNameList" >
            <list>
                <value>Core Java</value>
                <value>Design Patterns</value>
                <value>Advance Java</value>
                <value>Spring FrameWork</value>
                <value>Hibernate Framework</value>
            </list>
        </property>
    </bean>

    <bean id="customProcessor"
class="com.anurag.batch.processor.CustomItemProcessor" />

    <bean id="customWriter" class="com.anurag.batch.writer.CustomItemWriter" />

    <batch:job id="mySimpleJob" job-repository="jobRepository" parent="simpleJob">
        <batch:step id="step1">
            <batch:tasklet transaction-manager="transactionManager">
                <batch:chunk reader="customReader" processor="customProcessor"
                             writer="customWriter" commit-interval="1"/>
            </batch:tasklet>
        </batch:step>
    </batch:job>
</beans>
```

First we create three beans **(customReader, customWriter, ustomProcessor)** corresponding to **CustomItemReader**, **CustomItemWriter** and **CustomItemProcessor**. Note that the customReader is injected with the list of book names. This list is the source of data for the customReader bean.

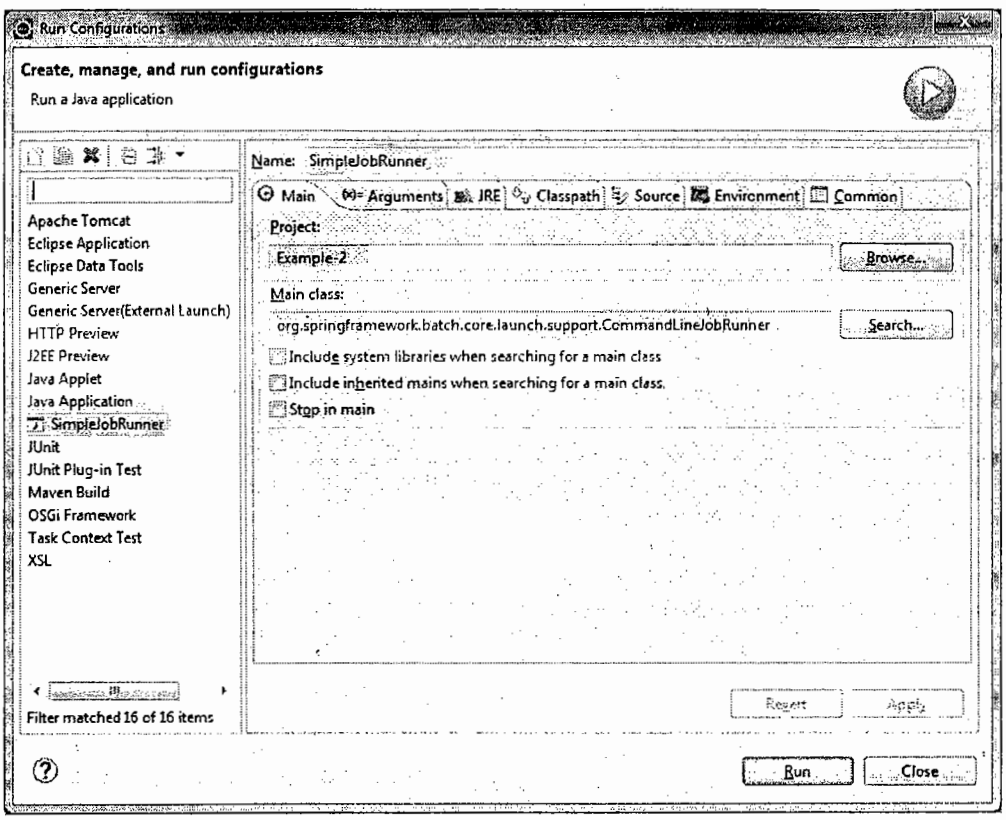
Later we created a simpleStep bean using the SimpleStepFactoryBean class. Most common configuration options for simple steps should be found in this factory class. We injected the jobRepository, transactionManager, customReader, customWriter and customProcessor in this simple step bean. Note the property commitInterval which is set to 1. This tells Spring Batch that the commit should happen after 1 element .i.e. writer will write 1 item at a time.

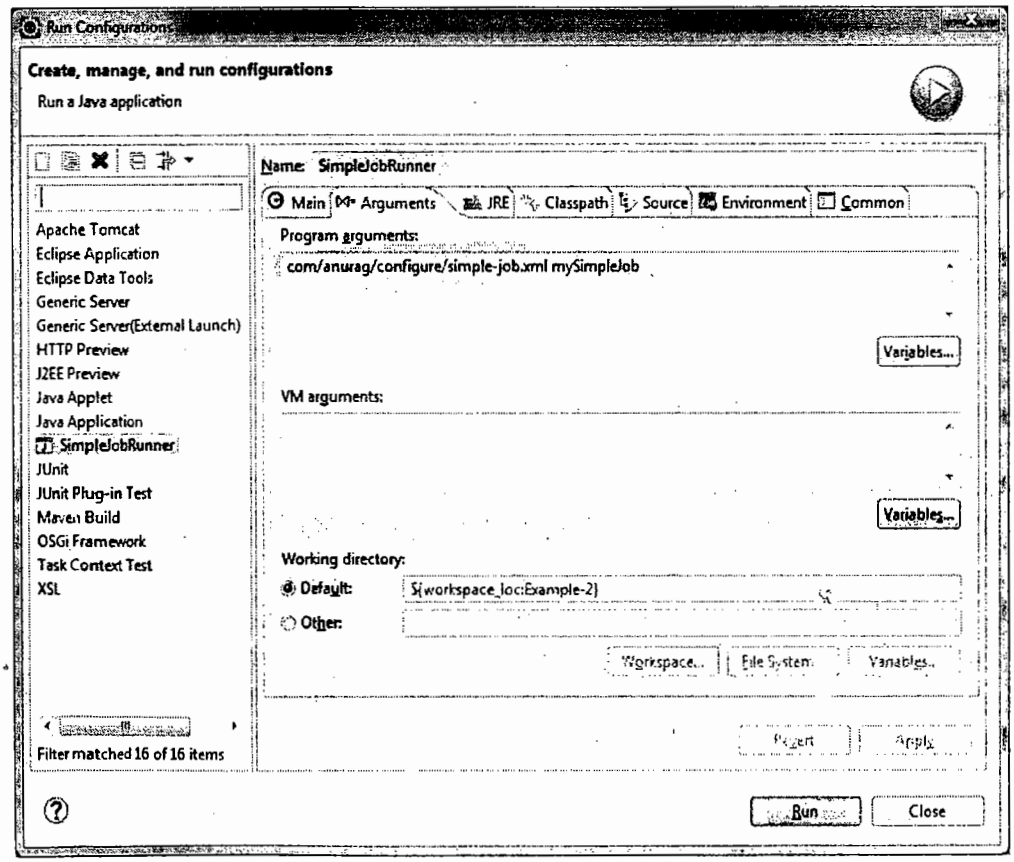
Launching Batch Job-

Spring Batch comes with a simple utility class called CommandLineJobRunner which has a main() method which accepts two arguments. First argument is the spring application context

file containing job definition and the second is the name of the job to be executed.

Now run as a java application with both two arguments.
org.springframework.batch.core.launch.support.CommandLineJobRunner simple-job.xml mySimpleJob





Following output-

```
<terminated> SimpleJobRunner [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Jun 23, 2016, 4:30:01 AM)
INFO: Loading XML bean definitions from class path resource [com/anurag/configure/applicationContext.xml]
Jun 23, 2016 4:30:02 AM org.springframework.beans.factory.support.DefaultListableBeanFactory registerBeanD
INFO: Overriding bean definition for bean 'mySimpleJob': replacing [Generic bean: class [org.springframewo
Jun 23, 2016 4:30:02 AM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiat
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
Jun 23, 2016 4:30:03 AM org.springframework.batch.core.launch.support.SimpleJobLauncher afterPropertiesSet
INFO: No TaskExecutor has been set, defaulting to synchronous executor.
Jun 23, 2016 4:30:03 AM org.springframework.batch.core.launch.support.SimpleJobLauncher$1 run
INFO: Job: [FlowJob: [name=mySimpleJob]] launched with the following parameters: [{}]
Jun 23, 2016 4:30:03 AM org.springframework.batch.core.job.AbstractJob handleStep
INFO: Executing step: [TaskletStep: [name=step1]]
[Book Name - Core Java | Author Name - Sriniwas]
[Book Name - Design Patterns | Author Name - Natraz Design Patterns]
[Book Name - Advance Java | Author Name - Natraz Adv. Java]
[Book Name - Spring FrameWork | Author Name - Natraz Spring]
[Book Name - Hibernate Framework | Author Name - Natraz Hibernate]
Jun 23, 2016 4:30:03 AM org.springframework.batch.core.launch.support.SimpleJobLauncher$1 run
INFO: Job: [FlowJob: [name=mySimpleJob]] completed with the following parameters: [{}] and the following s
Jun 23, 2016 4:30:03 AM org.springframework.context.support.AbstractApplicationContext doClose
INFO: Closing org.springframework.context.support.ClassPathXmlApplicationContext@1c12fb0: startup date [Th
Jun 23, 2016 4:30:03 AM org.springframework.beans.factory.support.DefaultSingletonBeanRegistry destroySing
INFO: Destroying singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@be92ad
```

Spring BOOT

Introducing Spring Boot

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

You can use Spring Boot to create Java applications that can be started using java -jar or more traditional war deployments. We also provide a command line tool that runs "spring scripts".

Our primary goals are:

- Provide a radically faster and widely accessible getting started experience for all Spring development.
- Be opinionated out of the box, but get out of the way quickly as requirements start to diverge from the defaults.
- Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration).
- Absolutely no code generation and no requirement for XML configuration.

System Requirements

By default, Spring Boot 2.0.0.BUILD-SNAPSHOT requires Java 7 and Spring Framework 5.0.0.BUILD-SNAPSHOT or above. You can use Spring Boot with Java 6 with some additional configuration. Explicit build support is provided for Maven (3.2+) and Gradle (1.12 or 2.x). Gradle 3 is not supported.

regular cfg of predefine classes will be care internally or through propertiesfiles
--Thumb rule:- Cgf user-defined classes through annotations,pre-defined classes through @bean method or through properties file

---: To evelop web application initializer in SpringBoot App take a class extending from SpringBootServletInitializer which takes cares DispatcherServlet initialization.
SpringBootServletInitializer
|----Internaly it is webApplicationinitializer.

Although you can use Spring Boot with Java 6 or 7, we generally recommend Java 8 if at all possible.

Servlet containers

The following embedded servlet containers are supported out of the box:

Name	Servlet Version	Java Version
Tomcat 8	3.1	Java 7+
Tomcat 7	3.0	Java 6+
Jetty 9.3	3.1	Java 8+
Jetty 9.2	3.1	Java 7+

Name	Servlet Version	Java Version
Jetty 8	3.0	Java 6+
Undertow 1.3	3.1	Java 7+

You can also deploy Spring Boot applications to any Servlet 3.0+ compatible container.

Installing Spring Boot

Spring Boot can be used with "classic" Java development tools or installed as a command line tool. Regardless, you will need Java SDK v1.6 or higher. You should check your current Java installation before you begin:

```
$ java -version
```

If you are new to Java development, or if you just want to experiment with Spring Boot you might want to try the Spring Boot CLI first, otherwise, read on for "classic" installation instructions.

Although Spring Boot is compatible with Java 1.6, if possible, you should consider using the latest version of Java.

Installation instructions for the Java developer

You can use Spring Boot in the same way as any standard Java library. Simply include the appropriate spring-boot-*.jar files on your classpath. Spring Boot does not require any special tools integration, so you can use any IDE or text editor; and there is nothing special about a Spring Boot application, so you can run and debug as you would any other Java program.

Although you *could* just copy Spring Boot jars, we generally recommend that you use a build tool that supports dependency management (such as Maven or Gradle).

Maven installation

Spring Boot is compatible with Apache Maven 3.2 or above. If you don't already have Maven installed you can follow the instructions at maven.apache.org.

On many operating systems Maven can be installed via a package manager. If you're an OSX Homebrew user try brew install maven. Ubuntu users can run sudo apt-get install maven.

Spring Boot dependencies use the org.springframework.boot groupId . Typically your Maven

POM file will inherit from the spring-boot-starter-parent project and declare dependencies to one or more "Starters". Spring Boot also provides an optional Maven plugin to create executable jars.

Here is a typical pom.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XM
LSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.example</groupId>
<artifactId>myproject</artifactId>
<version>0.0.1-SNAPSHOT</version>

<!-- Inherit defaults from Spring Boot -->
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.0.0.BUILD-SNAPSHOT</version>
</parent>

<!-- Add typical dependencies for a web application -->
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>

<!-- Package as an executable jar -->
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

<!-- Add Spring repositories -->
<!-- (you don't need this if you are using a .RELEASE version) -->
<repositories>
<repository>
<id>spring-snapshots</id>
<url>http://repo.spring.io/snapshot</url>
<snapshots><enabled>true</enabled></snapshots>
</repository>
<repository>
<id>spring-milestones</id>
<url>http://repo.spring.io/milestone</url>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
<id>spring-snapshots</id>
<url>http://repo.spring.io/snapshot</url>
```

launching Home page
procedure to develope spring application using spring boot::-
1- make sure that sts plugin install in eclipse
2-create spring starter project ----file-new-spring starter project-fill detail---select boot 1.4.7--search and select web---next--finish (gives 2 defaul classes webInitializer, configuration class)
3-add following dependency tag ---tomcat-embed-jasper
4-configure view resolver in application.properties as shown below
#VIEw Resolver Configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.sufix=.jsp
5-develop index.jsp and controller class regular manner
6-add @component scan in springbootConfiguration class specifing the packeg
7-run as spring boot application.

programmer runs application as spring boot application::---
1-main method of application class executes ---spring application.run method internally creat root(parrent) container by taking current class as configuration class beacuse of @springboot application (internal @configuration)---beacuse of that internal @enableautoconfiguration and pom.xml spring-boot-starter-tomcat jar embedded tomcat server will be started---that server serviceprovider class loader loads spring servletcontainerinitializer clas by loading it from meta-inf/ javax.servlet.ServletContainerInitializer file of spring/web/version.jar---that initializer takes our servletinitializer class that extends from spring boot servletinitializer ----by calling on startup method of spring boot servletinitializer it creates child container by t taking configuration method supplied configuration class(webmvcapplicationconfig)----attaches this container to dispathcer servlet as child container and registers dispatcherServlet servlet container dynamically with /url pattern . in this process the configure controler classand view resolver through properties file will be preintansitated and injections are completed
note:--parent/root container also complete preinstantiation and injection

```
</pluginRepository>
<pluginRepository>
<id>spring-milestones</id>
<url>http://repo.spring.io/milestone</url>
</pluginRepository>
</pluginRepositories>
</project>
```

while working with springboot app it looks to use DataSource in the following order
1-TomCat DataSource 2-hikari 3 apache dbcp 4-apache Dbcp2
to break above algorithm and to make spring boot giving container DataSource obj of our choice(belongs to use above) list directly.
along with regular jdbc DataSource cfg the following type property
spring.datasource.type=org.apache.commons.dbcp2.BasicDataSource

Generally it is recommended to use server managed jdbc connection pool in the application that are deployable in the server. Spring Boot Application we can mention datasource jndi in application.property as shown below.
"spring.datasource.jndi-name=java:/comp/env/Dsjndi"

The spring-boot-starter-parent is a great way to use Spring Boot, but it might not be suitable all of the time. Sometimes you may need to inherit from a different parent POM, or you might just not like our default settings. See Section 13.2.2, "Using Spring Boot without the parent POM" for an alternative solution that uses an import scope.

Developing your first Spring Boot application

Let's develop a simple "Hello World!" web application in Java that highlights some of Spring Boot's key features. We'll use Maven to build this project since most IDEs support it.

The spring.io web site contains many "Getting Started" guides that use Spring Boot. If you're looking to solve a specific problem; check there first.
You can shortcut the steps below by going to start.spring.io and choosing the web starter from the dependencies searcher. This will automatically generate a new project structure so that you can start coding right the way. Check the documentation for more details.

Before we begin, open a terminal to check that you have valid versions of Java and Maven installed.

```
$ java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)
$ mvn -v
Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8fdf4; 2014-08-11T13:58:10-07:00)
Maven home: /Users/user/tools/apache-maven-3.1.1
Java version: 1.7.0_51, vendor: Oracle Corporation
```

This sample needs to be created in its own folder. Subsequent instructions assume that you have created a suitable folder and that it is your "current directory".

Creating the POM

We need to start by creating a Maven pom.xml file. The pom.xml is the recipe that will be used to build your project. Open your favorite text editor and add the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchem
a-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>myproject</artifactId>
<version>0.0.1-SNAPSHOT</version>

<parent>
  <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.0.0.BUILD-SNAPSHOT</version>
</parent>

<!-- Additional lines to be added here... -->

<!-- (you don't need this if you are using a .RELEASE version) -->
<repositories>
  <repository>
   <id>spring-snapshots</id>
   <url>http://repo.spring.io/snapshot</url>
   <snapshots><enabled>true</enabled></snapshots>
 </repository>
 <repository>
   <id>spring-milestones</id>
    <url>http://repo.spring.io/milestone</url>
 </repository>
</repositories>
<pluginRepositories>
   <pluginRepository>
   <id>spring-snapshots</id>
    <url>http://repo.spring.io/snapshot</url>
</pluginRepository>
<pluginRepository>
   <id>spring-milestones</id>
   <url>http://repo.spring.io/milestone</url>
</pluginRepository>
</pluginRepositories>
</project>
```

This should give you a working build, you can test it out by running mvn package (you can ignore the "jar will be empty - no content was marked for inclusion!" warning for now).

while performing not select operation there is compatibility issue between spring 4.x and hibernate 5.x in orm module to over this problem use hibernate 4.x and also called ht.checkrwriteoperatio(false)

At this point you could import the project into an IDE (most modern Java IDE's include built-in support for Maven). For simplicity, we will continue to use a plain text editor for this example.

Adding classpath dependencies

Spring Boot provides a number of "Starters" that makes easy to add jars to your classpath. Our sample application has already used spring-boot-starter-parent in the parent section of the POM. The spring-boot-starter-parent is a special starter that provides useful Maven defaults. It also provides a dependency-management section so that you can omit version tags for "blessed" dependencies.

Other "Starters" simply provide dependencies that you are likely to need when developing a specific type of application. Since we are developing a web application, we will add a spring-boot-starter-web dependency — but before that, let's look at what we currently have.

```
$ mvn dependency:tree

[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT
```

The mvn dependency:tree command prints a tree representation of your project dependencies. You can see that spring-boot-starter-parent provides no dependencies by itself. Let's edit our pom.xml and add the spring-boot-starter-web dependency just below the parent section:

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
```

If you run mvn dependency:tree again, you will see that there are now a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

Writing the code

To finish our application we need to create a single Java file. Maven will compile sources from src/main/java by default so you need to create that folder structure, then add a file named src/main/java/Example.java :

```
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;
```

```
@RestController
@EnableAutoConfiguration
publicclass Example {

@RequestMapping("/")
   String home() {
return"Hello World!";
   }

publicstaticvoid main(String[] args) throws Exception {
      SpringApplication.run(Example.class, args);
   }
}
```

Although there isn't much code here, quite a lot is going on. Let's step through the important parts.

The @RestController and @RequestMapping annotations

The first annotation on our Example class is @RestController. This is known as a *stereotype* annotation. It provides hints for people reading the code, and for Spring, that the class plays a specific role. In this case, our class is a web @Controller so Spring will consider it when handling incoming web requests.

The @RequestMapping annotation provides "routing" information. It is telling Spring that any HTTP request with the path "/" should be mapped to the home method.
The @RestController annotation tells Spring to render the resulting string directly back to the caller.

The @RestController and @RequestMapping annotations are Spring MVC annotations (they are not specific to Spring Boot). See the MVC section in the Spring Reference Documentation for more details.

The @EnableAutoConfiguration annotation

The second class-level annotation is @EnableAutoConfiguration. This annotation tells Spring Boot to "guess" how you will want to configure Spring, based on the jar dependencies that you have added. Since spring-boot-starter-web added Tomcat and Spring MVC, the auto-configuration will assume that you are developing a web application and setup Spring accordingly.

Starters and Auto-Configuration

Auto-configuration is designed to work well with "Starters", but the two concepts are not directly tied. You are free to pick-and-choose jar dependencies outside of the starters and Spring Boot will still do its best to auto-configure your application.

The "main" method

The final part of our application is the `main` method. This is just a standard method that follows the Java convention for an application entry point. Our main method delegates to Spring Boot's `SpringApplication` class by calling `run`. `SpringApplication` will bootstrap our application, starting Spring which will in turn start the auto-configured Tomcat web server. We need to pass `Example.class` as an argument to the `run` method to tell `SpringApplication` which is the primary Spring component. The `args` array is also passed through to expose any command-line arguments.

Running the example

At this point our application should work. Since we have used the `spring-boot-starter-parent` POM we have a useful `run` goal that we can use to start the application. Type `mvn spring-boot:run` from the root project directory to start the application:

```
$ mvn spring-boot:run

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::  (v2.0.0.BUILD-SNAPSHOT)
....... . . .
....... . . . (log output here)
....... . . .
........ Started Example in 2.222 seconds (JVM running for 6.514)
```

If you open a web browser to localhost:8080 you should see the following output:

```
Hello World!
```

To gracefully exit the application hit `ctrl-c`.

11.5 Creating an executable jar

Let's finish our example by creating a completely self-contained executable jar file that we could run in production. Executable jars (sometimes called "fat jars") are archives containing your compiled classes along with all of the jar dependencies that your code needs to run.

Executable jars and Java

Java does not provide any standard way to load nested jar files (i.e. jar files that are themselves contained within a jar). This can be problematic if you are looking to distribute a self-contained application.

To solve this problem, many developers use "uber" jars. An uber jar simply packages all classes, from all jars, into a single archive. The problem with this approach is that it becomes hard to see which libraries you are actually using in your application. It can also be problematic if the same filename is used (but with different content) in multiple jars.

Spring Boot takes a different approach and allows you to actually nest jars directly.

To create an executable jar we need to add the spring-boot-maven-plugin to our pom.xml. Insert the following lines just below the dependencies section:

```
<build>
<plugins>
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>
```

The spring-boot-starter-parent POM includes <executions> configuration to bind the repackage goal. If you are not using the parent POM you will need to declare this configuration yourself. See the plugin documentation for details.

Save your pom.xml and run mvn package from the command line:

```
$ mvn package

[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building myproject 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO] .... ..
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myproject ---
[INFO] Building jar: /Users/developer/example/spring-boot-example/target/myproject-0.0.1-
SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.0.0.BUILD-SNAPSHOT:repackage (default) @ myproject
---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

If you look in the target directory you should see myproject-0.0.1-SNAPSHOT.jar. The file

should be around 10 Mb in size. If you want to peek inside, you can use `jar tvf`:

```
$ jar tvf target/myproject-0.0.1-SNAPSHOT.jar
```

You should also see a much smaller file named `myproject-0.0.1-SNAPSHOT.jar.original` in the `target` directory. This is the original jar file that Maven created before it was repackaged by Spring Boot.

To run that application, use the `java -jar` command:

```
$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::  (v2.0.0.BUILD-SNAPSHOT)
....... . . .
....... . . . (log output here)
....... . . .
........ Started Example in 2.536 seconds (JVM running for 2.864)
```

As before, to gracefully exit the application hit `ctrl-c`.

@@All the best my dear students@@

=======ooo=====

SPRING FAQs

Q. Explain DI or IOC pattern.

A: Dependency injection (DI) is a programming design pattern and architectural model, sometimes also referred to as inversion of control or IOC, although technically speaking, dependency injection specifically refers to an implementation of a particular form of IOC. Dependency Injection describes the situation where one object uses a second object to provide a particular capacity. For example, being passed a database connection as an argument to the constructor instead of creating one internally. The term "Dependency injection" is a misnomer, since it is not a dependency that is injected, rather it is a provider of some capability or resource that is injected. There are three common forms of dependency injection: setter-, constructor- and interface-based injection. Dependency injection is a way to achieve loose coupling. Inversion of control (IOC) relates to the way in which an object obtains references to its dependencies. This is often done by a lookup method. The advantage of inversion of control is that it decouples objects from specific lookup mechanisms and implementations of the objects it depends on. As a result, more flexibility is obtained for production applications as well as for testing.

Q. What are the different IOC containers available?

A. Spring is an IOC container. Other IOC containers are HiveMind, Avalon, PicoContainer.

Q. What are the different types of dependency injection. Explain with examples.

A: There are two types of dependency injection: setter injection and constructor injection. Setter Injection: Normally in all the java beans, we will use setter and getter method to set and get the value of property as follows:

```
public class namebean {
 String     name;
 public void setName(String a) {
    name = a; }
 public String getName() {
    return name; }
}
```

We will create an instance of the bean 'namebean' (say bean1) and set property as bean1.setName("tom"); Here in setter injection, we will set the property 'name' in spring configuration file as showm below:

```
<bean id="bean1"  class="namebean">
  <property  name="name" >
     <value>tom</value>
  </property>
</bean>
```

The subelement <value> sets the 'name' property by calling the set method as setName("tom"); This process is called setter injection.

To set properties that reference other beans <ref>, subelement of <property> is used as shown below,

```
<bean id="bean1"  class="bean1impl">
  <property name="game">
     <ref bean="bean2"/>
  </property>
</bean>
<bean id="bean2"  class="bean2impl" />
```

Constructor Injection: For constructor injection, we use constructor with parameters as shown below,

```
public class namebean {
   String name;
   public namebean(String a) {
      name = a;
   }
}
```

We will set the property 'name' while creating an instance of the bean 'namebean' as namebean bean1 = new namebean("tom");

Here we use the <constructor-arg> element to set the the property by constructor injection as

```
<bean id="bean1"  class="namebean">
  <constructor-arg>
     <value>My Bean Value</value>
  </constructor-arg>
</bean>
```

Q. What is spring? What are the various parts of spring framework? What are the different persistence frameworks which could be used with spring?

A. Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development. The Spring modules are built on top of the core container, which defines how beans are created, configured, and managed, as shown in the following figure. Each of the modules (or components) that comprise the Spring framework can stand on its own or be implemented jointly with one or more of the others. The functionality of each component is as follows:

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

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

Spring AOP: The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result

you can easily AOP-enable any object managed by the Spring framework. The Spring AOP module provides transaction management services for objects in any Spring-based application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

Spring DAO: The Spring JDBC DAO abstraction layer offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of exception code you need to write, such as opening and closing connections. Spring DAO's JDBC-oriented exceptions comply to its generic DAO exception hierarchy.

Spring ORM: The Spring framework plugs into several ORM frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps. All of these comply to Spring's generic transaction and DAO exception hierarchies.

Spring Web Module: The Web context module builds on top of the application context module, providing contexts for Web-based applications. As a result, the Spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

Spring MVC Framework: The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications. The MVC framework is highly configurable via strategy interfaces and accommodates numerous view technologies including JSP, Velocity, Tiles, iText, and POI.

Q. What is AOP? How does it relate with IOC? What are different tools to utilize AOP?

A: Aspect-oriented programming, or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. AOP and IOC are complementary technologies in that both apply a modular approach to complex problems in enterprise application development. In a typical object-oriented development approach you might implement logging functionality by putting logger statements in all your methods and Java classes. In an AOP approach you would instead modularize the logging services and apply them declaratively to the components that required logging. The advantage, of course, is that the Java class doesn't need to know about the existence of the logging service or concern itself with any related code. As a result, application code written using Spring AOP is loosely coupled. The best tool to utilize AOP to its capability is AspectJ. However AspectJ works at he byte code level and you need to use AspectJ compiler to get the aop features built into your compiled code. Nevertheless AOP functionality is fully integrated into the Spring context for transaction management, logging, and various other features. In general any AOP framework control aspects in three possible ways:

Joinpoints: Points in a program's execution. For example, joinpoints could define calls to specific methods in a class

Pointcuts: Program constructs to designate joinpoints and collect specific context at those points

Advices: Code that runs upon meeting certain conditions. For example, an advice could log a message before executing a joinpoint

Q. What are the advantages of spring framework?

A.

1. Spring has layed architecture. Use what you need and leave you don't need now.

2. Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables continous integration and testability.
3. Dependency Injection and Inversion of Control Simplifies JDBC (Read the first question.)
4. Open source and no vendor lock-in.

Q. Can you name a tool which could provide the initial ant files and directory structure for a new spring project.
A: Appfuse or equinox.

Q. Explain BeanFactory in spring.
A: Bean factory is an implementation of the factory design pattern and its function is to create and dispense beans. As the bean factory knows about many objects within an application, it is able to create association between collaborating objects as they are instantiated. This removes the burden of configuration from the bean and the client. There are several implementation of BeanFactory. The most useful one is "org.springframework.beans.factory.xml.XmlBeanFactory" It loads its beans based on the definition contained in an XML file. To create an XmlBeanFactory, pass a InputStream to the constructor. The resource will provide the XML to the factory. BeanFactory factory = new XmlBeanFactory(new FileInputStream("myBean.xml"));

This line tells the bean factory to read the bean definition from the XML file. The bean definition includes the description of beans and their properties. But the bean factory doesn't instantiate the bean yet. To retrieve a bean from a 'BeanFactory', the getBean() method is called. When getBean() method is called, factory will instantiate the bean and begin setting the bean's properties using dependency injection. myBean bean1 = (myBean)factory.getBean("myBean");

Q. Explain the role of ApplicationContext in spring.
A. While Bean Factory is used for simple applications, the Application Context is spring's more advanced container. Like 'BeanFactory' it can be used to load bean definitions, wire beans together and dispense beans upon request. It also provide
1) a means for resolving text messages, including support for internationalization.
2) a generic way to load file resources.
3) events to beans that are registered as listeners.
Because of additional functionality, 'Application Context' is preferred over a BeanFactory. Only when the resource is scarce like mobile devices, 'BeanFactory' is used. The three commonly used implementation of 'Application Context' are
1. ClassPathXmlApplicationContext : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code
ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
2. FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code
ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
3. XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

Q. How does Spring supports DAO in hibernate?
A. Spring's HibernateDaoSupport class is a convenient super class for Hibernate DAOs. It has handy methods you can call to get a Hibernate Session, or a SessionFactory. The most

convenient method is getHibernateTemplate(), which returns a HibernateTemplate. This template wraps Hibernate checked exceptions with runtime exceptions, allowing your DAO interfaces to be Hibernate exception-free.

Example:

```
public class UserDAOHibernate extends HibernateDaoSupport {

public User getUser(Long id) {
return (User) getHibernateTemplate().get(User.class, id);
}
public void saveUser(User user) {
getHibernateTemplate().saveOrUpdate(user);
if (log.isDebugEnabled()) {
log.debug("userId set to: " + user.getID());
}
}
public void removeUser(Long id) {
Object user = getHibernateTemplate().load(User.class, id);
getHibernateTemplate().delete(user);
}
}
```

Q. What are the id generator classes in hibernate?

A: increment: It generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. It should not the used in the clustered environment.

identity: It supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL. The returned identifier is of type long, short or int.

sequence: The sequence generator uses a sequence in DB2, PostgreSQL, Oracle, SAP Database, McKoi or a generator in Interbase. The returned identifier is of type long, short or int

hilo: The hilo generator uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a table and column (by default hibernate_unique_key and next_hi respectively) as a source of hi values. The hi/lo algorithm generates identifiers that are unique only for a particular database. Do not use this generator with connections enlisted with JTA or with a user-supplied connection.

seqhilo: The seqhilo generator uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a named database sequence.

uuid: The uuid generator uses a 128-bit UUID algorithm to generate identifiers of type String, unique within a network (the IP address is used). The UUID is encoded as a String of hexadecimal digits of length 32.

guid: It uses a database-generated GUID String on MS SQL Server and MySQL.

native: It picks identity, sequence or hilo depending upon the capabilities of the underlying database.

assigned: lets the application to assign an identifier to the object before save() is called. This is the default strategy if no <generator> element is specified.

select: retrieves a primary key assigned by a database trigger by selecting the row by some unique key and retrieving the primary key value.

foreign: uses the identifier of another associated object. Usually used in conjunction with a <one-to-one> primary key association.

Q. How is a typical spring implementation look like?

A. For a typical Spring Application we need the following files

1. An interface that defines the functions.
2. An Implementation that contains properties, its setter and getter methods, functions etc.,
3. A XML file called Spring configuration file.
4. Client program that uses the function.

Q. How do you define hibernate mapping file in spring?

A. Add the hibernate mapping file entry in mapping resource inside Spring's applicationContext.xml file in the web/WEB-INF directory.

```
<property name="mappingResources">
  <list>
    <value>org/appfuse/model/User.hbm.xml</value>
  </list>
</property>
```

Q. How do you configure spring in a web application?

A. It is very easy to configure any J2EE-based web application to use Spring. At the very least, you can simply add Spring's ContextLoaderListener to your web.xml file:

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

Q. Can you have xyz.xml file instead of applicationContext.xml?

A. ContextLoaderListener is a ServletContextListener that initializes when your webapp starts up. By default, it looks for Spring's configuration file at WEB-INF/applicationContext.xml. You can change this default value by specifying a <context-param> element named "contextConfigLocation." Example:

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

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/xyz.xml</param-value>
  </context-param>
  </listener-class>
</listener>
```

Q. How do you configure your database driver in spring?

A. Using datasource "org.springframework.jdbc.datasource.DriverManagerDataSource".

Example:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName">
    <value>org.hsqldb.jdbcDriver</value>
  </property>
  <property name="url">
    <value>jdbc:hsqldb:db/appfuse</value>
  </property>
```

```
    <property name="username">><value>sa</value></property>
    <property name="password">><value></value></property>
</bean>
```

Q. How can you configure JNDI instead of datasource in spring applicationContext.xml?

A. Using "org.springframework.jndi.JndiObjectFactoryBean". Example:

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
        <value>java:comp/env/jdbc/appfuse</value>
    </property>
</bean>
```

Q. What are the key benefits of Hibernate?

A: These are the key benefits of Hibernate:

- Transparent persistence based on POJOs without byte code processing
- Powerful object-oriented hibernate query language
- Descriptive O/R Mapping through mapping file.
- Automatic primary key generation
- Hibernate cache : Session Level, Query and Second level cache.
- Performance: Lazy initialization, Outer join fetching, Batch fetching

Q. What is hibernate session and session factory? How do you configure sessionfactory in spring configuration file?

A. Hibernate Session is the main runtime interface between a Java application and Hibernate. SessionFactory allows applications to create hibernate session by reading hibernate configurations file hibernate.configure.xml.

```
// Initialize the Hibernate environment
Configuration configure = new Configuration().configure();
// Create the session factory
SessionFactory factory = configure.buildSessionFactory();
// Obtain the new session object
Session session = factory.openSession();
```

The call to Configuration().configure() loads the hibernate.configure.xml configuration file and initializes the Hibernate environment. Once the configuration is initialized, you can make any additional modifications you desire programmatically. However, you must make these modifications prior to creating the SessionFactory instance. An instance of SessionFactory is typically created once and used to create all sessions related to a given context.

The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of three states:

transient: never persistent, not associated with any Session

persistent: associated with a unique Session

detached: previously persistent, not associated with any Session

A Hibernate Session object represents a single unit-of-work for a given data store and is opened by a SessionFactory instance. You must close Sessions when all work for a transaction is completed. The following illustrates a typical Hibernate session:

```
Session session = null;
UserInfo user = null;
```

```
Transaction tx = null;
try {
  session = factory.openSession();
  tx = session.beginTransaction();
  user = (UserInfo)session.load(UserInfo.class, id);
  tx.commit();
} catch(Exception e) {
  if (tx != null) {
    try {
      tx.rollback();
    } catch (HibernateException e1) {
      throw new DAOException(e1.toString()); }
  } throw new DAOException(e.toString());
} finally {
  if (session != null) {
    try {
      session.close();
    } catch (HibernateException e) { }
  }
}
```

Q. What is the difference between hibernate get and load methods?

A. The load() method is older; get() was added to Hibernate's API due to user request. The difference is trivial:

The following Hibernate code snippet retrieves a User object from the database: User user = (User) session.get(User.class, userID);

The get() method is special because the identifier uniquely identifies a single instance of a class. Hence it's common for applications to use the identifier as a convenient handle to a persistent object. Retrieval by identifier can use the cache when retrieving an object, avoiding a database hit if the object is already cached.

Hibernate also provides a load() method: User user = (User) session.load(User.class, userID); If load() can't find the object in the cache or database, an exception is thrown. The load() method never returns null. The get() method returns null if the object can't be found. The load() method may return a proxy instead of a real persistent instance. A proxy is a placeholder instance of a runtime-generated subclass (through cglib or Javassist) of a mapped persistent class, it can initialize itself if any method is called that is not the mapped database identifier getter-method. On the other hand, get() never returns a proxy. Choosing between get() and load() is easy: If you're certain the persistent object exists, and nonexistence would be considered exceptional, load() is a good option. If you aren't certain there is a persistent instance with the given identifier, use get() and test the return value to see if it's null. Using load() has a further implication: The application may retrieve a valid reference (a proxy) to a persistent instance without hitting the database to retrieve its persistent state. So load() might not throw an exception when it doesn't find the persistent object in the cache or database; the exception would be thrown later, when the proxy is accessed.

Q. What type of transaction management is supported in hibernate?

A. Hibernate communicates with the database via a JDBC Connection; hence it must support both managed and non-managed transactions.

non-managed in web containers:

```
<bean id="transactionManager"
```

```
class="org.springframework.orm.hibernate.HibernateTransactionManager">
   <property name="sessionFactory">
      <ref local="sessionFactory"/>
   </property>
</bean>
```

managed in application server using JTA:

```
 <bean id="transactionManager"
class="org.springframework.transaction.jta.JtaTransactionManager.">
   <property name="sessionFactory">
      <ref local="sessionFactory"/>
   </property>
</bean>
```

Q. What is lazy loading and how do you achieve that in hibernate?

A. Lazy setting decides whether to load child objects while loading the Parent Object. You need to specify parent class.Lazy = true in hibernate mapping file. By default the lazy loading of the child objects is true. This make sure that the child objects are not loaded unless they are explicitly invoked in the application by calling getChild() method on parent. In this case hibernate issues a fresh database call to load the child when getChild() is actully called on the Parent object. But in some cases you do need to load the child objects when parent is loaded. Just make the lazy=false and hibernate will load the child when parent is loaded from the database. Examples:- Address child of User class can be made lazy if it is not required frequently. But you may need to load the Author object for Book parent whenever you deal with the book for online bookshop.

Hibernate does not support lazy initialization for detached objects. Access to a lazy association outside of the context of an open Hibernate session will result in an exception.

Q. What are the different fetching strategy in Hibernate?

A. Hibernate3 defines the following fetching strategies:

Join fetching - Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.

Select fetching - a second SELECT is used to retrieve the associated entity or collection. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.

Subselect fetching - a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association. Batch fetching - an optimization strategy for select fetching - Hibernate retrieves a batch of entity instances or collections in a single SELECT, by specifying a list of primary keys or foreign keys.

For more details read short primer on fetching strategy at http://www.hibernate.org/315.html

Q. What are different types of cache hibernate supports ?

A. Caching is widely used for optimizing database applications. Hibernate uses two different caches for objects: first-level cache and second-level cache. First-level cache is associated with the Session object, while second-level cache is associated with the Session Factory object. By default, Hibernate uses first-level cache on a per-transaction basis. Hibernate uses this cache mainly to reduce the number of SQL queries it needs to generate within a given transaction. For

example, if an object is modified several times within the same transaction, Hibernate will generate only one SQL UPDATE statement at the end of the transaction, containing all the modifications. To reduce database traffic, second-level cache keeps loaded objects at the Session Factory level between transactions. These objects are available to the whole application, not just to the user running the query. This way, each time a query returns an object that is already loaded in the cache, one or more database transactions potentially are avoided. In addition, you can use a query-level cache if you need to cache actual query results, rather than just persistent objects. The query cache should always be used in conjunction with the second-level cache. Hibernate supports the following open-source cache implementations out-of-the-box:

- **EHCache** is a fast, lightweight, and easy-to-use in-process cache. It supports read-only and read/write caching, and memory- and disk-based caching. However, it does not support clustering.
- **OSCache** is another open-source caching solution. It is part of a larger package, which also provides caching functionalities for JSP pages or arbitrary objects. It is a powerful and flexible package, which, like EHCache, supports read-only and read/write caching, and memory- and disk-based caching. It also provides basic support for clustering via either JavaGroups or JMS.
- **SwarmCache** is a simple cluster-based caching solution based on JavaGroups. It supports read-only or nonstrict read/write caching (the next section explains this term). This type of cache is appropriate for applications that typically have many more read operations than write operations.
- **JBoss TreeCache** is a powerful replicated (synchronous or asynchronous) and transactional cache. Use this solution if you really need a true transaction-capable caching architecture.
- Commercial Tangosol Coherence cache.

Q. What are the different caching strategies?

A. The following four caching strategies are available:

- Read-only: This strategy is useful for data that is read frequently but never updated. This is by far the simplest and best-performing cache strategy.
- Read/write: Read/write caches may be appropriate if your data needs to be updated. They carry more overhead than read-only caches. In non-JTA environments, each transaction should be completed when Session.close() or Session.disconnect() is called.
- Nonstrict read/write: This strategy does not guarantee that two transactions won't simultaneously modify the same data. Therefore, it may be most appropriate for data that is read often but only occasionally modified.
- Transactional: This is a fully transactional cache that may be used only in a JTA environment.

Q. How do you configure 2nd level cach in hibernate?

A. To activate second-level caching, you need to define the hibernate.cache.provider_class property in the hibernate.configure.xml file as follows: <hibernate-configuration>

```
<session-factory>
    <property
name="hibernate.cache.provider_class">org.hibernate.cache.EHCacheProvider</property>
  </session-factory>
</hibernate-configuration>
```

By default, the second-level cache is activated and uses the EHCache provider.
To use the query cache you must first enable it by setting the property hibernate.cache.use_query_cache to true in hibernate.properties.

Q. What is the difference between sorted and ordered collection in hibernate?
A. A sorted collection is sorted in-memory using java comparator, while order collection is ordered at the database level using order by clause.

Q. What are the types of inheritence models and describe how they work like vertical inheritence and horizontal?
A. There are three types of inheritance mapping in hibernate :

Example: Let us take the simple example of 3 java classes. Class Manager and Worker are inherited from Employee Abstract class.
1. Table per concrete class with unions : In this case there will be 2 tables. Tables: Manager, Worker [all common attributes will be duplicated]
2. Table per class hierarchy: Single Table can be mapped to a class hierarchy. There will be only one table in database called 'Employee' that will represent all the attributes required for all 3 classes. But it needs some discriminating column to differentiate between Manager and worker;
3. Table per subclass: In this case there will be 3 tables represent Employee, Manager and Worker

Additional Questions:

1. What is IOC (or Dependency Injection)?
The basic concept of the Inversion of Control pattern (also known as dependency injection) is that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (in the case of the Spring framework, the IOC container) is then responsible for hooking it all up.
i.e., Applying IoC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is, dependencies are injected into objects. So, IoC means an inversion of responsibility with regard to how an object obtains references to collaborating objects.

2. What are the different types of IOC (dependency injection) ?
There are three types of dependency injection:

- Constructor Injection (e.g. Pico container, Spring etc): Dependencies are provided as constructor parameters.
- Setter Injection (e.g. Spring): Dependencies are assigned through JavaBeans properties (eg: setter methods).
- Interface Injection (e.g. Avalon): Injection is done through an interface.

Note: Spring supports only Constructor and Setter Injection

3. What are the benefits of IOC (Dependency Injection)?
Benefits of IOC (Dependency Injection) are as follows:

- Minimizes the amount of code in your application. With IOC containers you do not care about how services are created and how you get references to the ones you need. You can also easily add additional services by adding a new constructor or a setter method

with little or no extra configuration.

- Make your application more testable by not requiring any singletons or JNDI lookup mechanisms in your unit test cases. IOC containers make unit testing and switching implementations very easy by manually allowing you to inject your own objects into the object under test.
- Loose coupling is promoted with minimal effort and least intrusive mechanism. The factory design pattern is more intrusive because components or services need to be requested explicitly whereas in IOC the dependency is injected into requesting piece of code. Also some containers promote the design to interfaces not to implementations design concept by encouraging managed objects to implement a well-defined service interface of your own.
- IOC containers support eager instantiation and lazy loading of services. Containers also provide support for instantiation of managed objects, cyclical dependencies, life cycles management, and dependency resolution between managed objects etc.

4. What is Spring ?

Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

5. What are the advantages of Spring framework?

The advantages of Spring are as follows:

- Spring has layered architecture. Use what you need and leave you don't need now.
- Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables continuous integration and testability.
- Dependency Injection and Inversion of Control Simplifies JDBC
- Open source and no vendor lock-in.

6. What are features of Spring ?

Lightweight:

spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB. And the processing overhead is also very negligible.

Inversion of control (IOC):

Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.

Aspect oriented (AOP):

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

Container:

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

MVC Framework:

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

Transaction Management:

Spring framework provides a generic abstraction layer for transaction management. This

allowing the developer to add the pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.

JDBC Exception Handling:

The JDBC abstraction layer of the Spring offers a meaningful exception hierarchy, which simplifies the error handling strategy. Integration with Hibernate, JDO, and iBATIS: Spring provides best Integration services with Hibernate, JDO and iBATIS

contexts for Web-based applications. As a result, the Spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

7 . What is web module?

This module is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with Jakarta Struts.

8. What are the types of Dependency Injection Spring supports?

Setter Injection:

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

Constructor Injection:

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

9. What is Bean Factory ?

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

- BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.
- BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

10. What is Application Context?

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

- A means for resolving text messages, including support for internationalization.
- A generic way to load file resources.
- Events to beans that are registered as listeners

11. What is the difference between Bean Factory and Application Context ?

On the surface, an application context is same as a bean factory. But application context offers much more..

- Application contexts provide a means for resolving text messages, including support for

i18n of those messages.

- Application contexts provide a generic way to load file resources, such as images.
- Application contexts can publish events to beans that are registered as listeners.
- Certain operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context.
- ResourceLoader support: Spring's Resource interface us a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader, Hence provides an application with access to deployment-specific Resource instances.
- MessageSource support: The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable

12. What are the common implementations of the Application Context ?

The three commonly used implementation of 'Application Context' are

- ClassPathXmlApplicationContext : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code .
- ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
- FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code .
- ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
- XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

13. How is a typical spring implementation look like ?

For a typical Spring Application we need the following files:

- An interface that defines the functions.
- An Implementation that contains properties, its setter and getter methods, functions etc.,
- Spring AOP (Aspect Oriented Programming)
- A XML file called Spring configuration file.
- Client program that uses the function.

14. What is the typical Bean life cycle in Spring Bean Factory Container ?

Bean life cycle in Spring Bean Factory Container is as follows:

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

15. What do you mean by Bean wiring ?

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

16. What do you mean by Auto Wiring?

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

- no
- byName
- byType
- constructor
- autodirect

17. What is DelegatingVariableResolver?

Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard Java Server Faces managed beans mechanism which lets you use JSPRINGFRAMEWORK and Spring together. This variable resolver is called as DelegatingVariableResolver

18. How to integrate Java Server Faces (JSPRINGFRAMEWORK) with Spring?

JSPRINGFRAMEWORK and Spring do share some of the same features, most noticeably in the area of IOC services. By declaring JSPRINGFRAMEWORK managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSPRINGFRAMEWORK pages have access to these beans and all of their properties.We can integrate JSPRINGFRAMEWORK and Spring in two ways:

- DelegatingVariableResolver: Spring comes with a JSPRINGFRAMEWORK variable resolver that lets you use JSPRINGFRAMEWORK and Spring together.

```
<?xml version="1.0" encoding="UTF-8"?>
< !DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<faces-config>
<application>
<variable-resolver>
        org.springframework.web.jspringframework.DelegatingVariableResolver
</variable-resolver>
</application>
< /faces-config>
```

The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSPRINGFRAMEWORK implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSPRINGFRAMEWORK-managed beans.

- FacesContextUtils:custom VariableResolver works well when mapping one's properties to beans in faces-config.xml, but at times one may need to grab a bean explicitly. The FacesContextUtils class makes this easy. It is similar to WebApplicationContextUtils, except that it takes a FacesContext parameter rather than a ServletContext parameter.

```
ApplicationContext ctx =
FacesContextUtils.getWebApplicationContext(FacesContext.getCurrentInstance());
```

19. What is Java Server Faces (JSPRINGFRAMEWORK) - Spring integration mechanism?

Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard JavaServer Faces managed beans mechanism. When asked to resolve a variable name, the following algorithm is performed:

- Does a bean with the specified name already exist in some scope (request, session, application)? If so, return it
- Is there a standard JavaServer Faces managed bean definition for this variable name? If so, invoke it in the usual way, and return the bean that was created.
- Is there configuration information for this variable name in the Spring WebApplicationContext for this application? If so, use it to create and configure an instance, and return that instance to the caller.
- If there is no managed bean or Spring definition for this variable name, return null instead.
- BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

As a result of this algorithm, you can transparently use either JavaServer Faces or Spring facilities to create beans on demand.

20. What is Significance of JSPRINGFRAMEWORK- Spring integration ?

Spring - JSPRINGFRAMEWORK integration is useful when an event handler wishes to explicitly invoke the bean factory to create beans on demand, such as a bean that encapsulates the business logic to be performed when a submit button is pressed.

21. How to integrate your Struts application with Spring?

To integrate your Struts application with Spring, we have two options:

- Configure Spring to manage your Actions as beans, using the ContextLoaderPlugin, and set their dependencies in a Spring context file.
- Subclass Spring's ActionSupport classes and grab your Spring-managed beans explicitly using a getWebApplicationContext() method.

22. What are ORM's Spring supports ?

Spring supports the following ORM's :

- Hibernate
- iBatis
- JPA (Java Persistence API)
- TopLink
- JDO (Java Data Objects)
- OJB

23. What are the ways to access Hibernate using Spring ?

There are two approaches to Spring's Hibernate integration:

- Inversion of Control with a HibernateTemplate and Callback
- Extending HibernateDaoSupport and Applying an AOP Interceptor

24. How to integrate Spring and Hibernate using HibernateDaoSupport?

Spring and Hibernate can integrate using Spring's SessionFactory called LocalSessionFactory. The integration process is of 3 steps.

- Configure the Hibernate SessionFactory

- Extend your DAO Implementation from HibernateDaoSupport
- Wire in Transaction Support with AOP

25. What are Bean scopes in Spring Framework ?

The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext). The scopes supported are listed below:

singleton

Scopes a single bean definition to a single object instance per Spring IoC container.

prototype

Scopes a single bean definition to any number of object instances.

request

Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.

session

Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.

global session

Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

26. What is AOP?

Aspect-oriented programming, or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules.

27. How the AOP used in Spring?

AOP is used in the Spring Framework: To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on the Spring Framework's transaction abstraction. To allow users to implement custom aspects, complementing their use of OOP with AOP.

28. What do you mean by Aspect ?

A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

29. What do you mean by JointPoint?

A point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.

30. What do you mean by Advice?

Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors "around" the join point.

31. What are the types of Advice?

Types of advice:

- Before advice: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
- After returning advice: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
- After throwing advice: Advice to be executed if a method exits by throwing an exception.
- After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
- Around advice: Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception

32. What are the types of the transaction management Spring supports ?

Spring Framework supports:

- Programmatic transaction management.
- Declarative transaction management.

33. What are the benefits of the Spring Framework transaction management ?

The Spring Framework provides a consistent abstraction for transaction management that delivers the following benefits:

- Provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
- Supports declarative transaction management.
- Provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
- Integrates very well with Spring's various data access abstractions.

34. Why most users of the Spring Framework choose declarative transaction management ?

Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container.

35. Explain the similarities and differences between EJB CMT and the Spring Framework's declarative transaction management?

The basic approach is similar: it is possible to specify transaction behavior (or lack of it) down to individual method level. It is
possible to make a setRollbackOnly() call within a transaction context if necessary. The differences are:

- Unlike EJB CMT, which is tied to JTA, the Spring Framework's declarative transaction management works in any environment. It can work with JDBC, JDO, Hibernate or other transactions under the covers, with configuration changes only.
- The Spring Framework enables declarative transaction management to be applied to any class, not merely special classes such as EJBs.
- The Spring Framework offers declarative rollback rules: this is a feature with no EJB equivalent. Both programmatic and declarative support for rollback rules is provided.

- The Spring Framework gives you an opportunity to customize transactional behavior, using AOP. With EJB CMT, you have no way to influence the container's transaction management other than setRollbackOnly().
- The Spring Framework does not support propagation of transaction contexts across remote calls, as do high-end application servers.

36. What are object/relational mapping integration module?

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

37. When to use programmatic and declarative transaction management ?

Programmatic transaction management is usually a good idea only if you have a small number of transactional operations. On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure.

38. Explain about the Spring DAO support ?

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

39. What are the exceptions thrown by the Spring DAO classes ?

Spring DAO classes throw exceptions which are subclasses of DataAccessException(org.springframework.dao.DataAccessException).Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.

40. What is SQLExceptionTranslator ?

SQLExceptionTranslator, is an interface to be implemented by classes that can translate between SQLExceptions and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

41. What is Spring's JdbcTemplate ?

Spring's JdbcTemplate is central class to interact with a database through JDBC. JdbcTemplate provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

```
JdbcTemplate template = new JdbcTemplate(myDataSource);
```

42. What is PreparedStatementCreator ?

PreparedStatementCreator:

- Is one of the most common used interfaces for writing data to database.
- Has one method – createPreparedStatement(Connection)
- Responsible for creating a PreparedStatement.
- Does not need to handle SQLExceptions.

43. What is SQLProvider ?

SQLProvider:

- Has one method – getSql()
- Typically implemented by PreparedStatementCreator implementers.
- Useful for debugging.

44. What is RowCallbackHandler ?

The RowCallbackHandler interface extracts values from each row of a ResultSet.

- Has one method – processRow(ResultSet)
- Called for each row in ResultSet.
- Typically stateful.

45. What are the differences between EJB and Spring ?

Transaction management

EJB:

- Must use a JTA transaction manager.
- Supports transactions that span remote method calls.

Spring:

- Supports multiple transaction environments through its PlatformTransactionManager interface, including JTA, Hibernate, JDO, and JDBC.
- Does not natively support distributed transactions—it must be used with a JTA transaction manager.

Declarative transaction support

EJB:

- Can define transactions declaratively through the deployment descriptor.
- Can define transaction behavior per method or per class by using the wildcard character *.
- Cannot declaratively define rollback behavior—this must be done programmatically.

Spring:

- Can define transactions declaratively through the Spring configuration file or through class metadata.
- Can define which methods to apply transaction behavior explicitly or by using regular expressions.
- Can declaratively define rollback behavior per method and per exception type.

Persistence

EJB:

Supports programmatic bean-managed persistence and declarative container managed persistence.

Spring:

Provides a framework for integrating with several persistence technologies, including JDBC, Hibernate, JDO, and iBATIS.

Declarative security

EJB:

- Supports declarative security through users and roles. The management and implementation of users and roles is container specific.
- Declarative security is configured in the deployment descriptor.

Spring:

- No security implementation out-of-the box.
- Acegi, an open source security framework built on top of Spring, provides declarative

security through the Spring configuration file or class metadata.

Distributed computing

EJB:

Provides container-managed remote method calls.

Spring:

Provides proxying for remote calls via RMI, JAX-RPC, and web services.

46 . Do I need any other SOAP framework to run Spring Web Services?

You don't need any other SOAP framework to use Spring Web services, though it can use some of the features of Axis 1 and 2.

47 . I get NAMESPACE_ERR exceptions when using Spring-WS. What can I do about it?

If you get the following Exception:

NAMESPACE_ERR: An attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

Most often, this exception is related to an older version of Xalan being used. Make sure to upgrade to 2.7.0.

48 .Does Spring-WS run under Java 1.3?

Spring Web Services requires Java 1.4 or higher.

49. Does Spring-WS work under Java 1.4?

Spring Web Services works under Java 1.4, but it requires some effort to make it work. Java 1.4 is bundled with the older XML parser Crimson, which does not handle namespaces correctly. Additionally, it is bundled with an older version of Xalan, which also has problems. Unfortunately, placing newer versions of these on the class path does not override them. . The only solution that works is to add newer versions of Xerces and Xalan in the lib/endorsed directory of your JDK, as explained in those FAQs (i.e.$JAVA_HOME/lib/endorsed). The following libraries are known to work with Java 1.4.2:

Library	Version
Xerces	2.8.1
Xalan	2.7.0
XML-APIs	1.3.04
SAAJ	1.2

If you want to use WS-Security, note that the XwsSecurityInterceptor requires Java 5, because an underlying library (XWSS) requires it. Instead, you can use the Wss4jSecurityInterceptor.

50 .Does Spring-WS work under Java 1.6?

Java 1.6 ships with SAAJ 1.3, JAXB 2.0, and JAXP 1.4 (a custom version of Xerces and Xalan). Overriding these libraries by putting different version on the classpath will result in various classloading issues, or exceptions in org.apache.xml.serializer.ToXMLSAXHandler. The only option for using more recent versions is to put the newer version in the endorsed directory (see above).

51 . Why do the Spring-WS unit tests fail under Mac OS X?

For some reason, Apple decided to include a Java 1.4 compatibility jar with their JDK 1.5. This jar includes the XML parsers which were included in Java 1.4. No other JDK distribution does this, so it is unclear what the purpose of this compatibility jar is.

The jar can be found at
/System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Classes/.compatibility/14compatibility.jar. You can safely remove or rename it, and the tests will run again.

52 . What is SAAJ?
SAAJ is the SOAP with Attachments API for Java. Like most Java EE libraries, it consists of a set of interfaces (saaj-api.jar), and implementations (saaj-impl.jar). When running in a Application Server, the implementation is typically provided by the application server. Previously, SAAJ has been part of JAXM, but it has been released as a seperate API as part of the , and also as part of J2EE 1.4. SAAJ is generally known as the packagejavax.xml.soap.
Spring-WS uses this standard SAAJ library to create representations of SOAP messages. Alternatively, it can use

59. Why does Spring-WS only support contract-first?
Note that Spring-WS only requires you to write the XSD; the WSDL can be generated from that.

60 . How do I retrieve the WSDL from a Service?
The &WSDL query parameter does not work.
The &WSDL query parameter is a way to get a WSDL of a class. In SWS, a service is generally not implemented as a single class, but as a collection of endpoints.
There are two ways to expose a WSDL:

- Simply add the WSDL to the root of the WAR, and the file is served normally. This has the disadvantage that the "location" attribute in the WSDL is static, i.e. it does not necessarily reflect the host name of the server. You can transpringframeworkorm locations by using a WsdlDefinitionHandlerAdapter.
- Use theMessageDispatcherServlet, which is done is the samples. Every WsdlDefinition listed in the *-servlet.xml will be exposed under the bean name. So if you define a WsdlDefinition namedecho, it will be exposed as echo.wsdl (i.e.http://localhost:8080/echo/echo.wsdl).

61 What is web module?
Spring comes with a full-featured MVC framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring's MVC framework uses IoC to provide for a clean separation of controller logic from business objects. It also allows you to declaratively bind request parameters to your business objects. It also can take advantage of any of Spring's other services, such as I18N messaging and validation.

62 What is a BeanFactory?
A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application's configuration and dependencies from the actual application code.

63 What is AOP Alliance?
AOP Alliance is an open-source project whose goal is to promote adoption of AOP and interoperability among different AOP implementations by defining a common set of interfaces and components.

64 What is Spring configuration file?
Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

65 .What does a simple spring application contain?

These applications are like any Java application. They are made up of several classes, each performing a specific purpose within the application. But these classes are configured and introduced to each other through an XML file. This XML file describes how to configure the classes, known as the Spring configuration file.

66. What is XMLBeanFactory?

BeanFactory has many implementations in Spring. But one of the most useful one is org.springframework.beans.factory.xml.XmlBeanFactory, which loads its beans based on the definitions contained in an XML file. To create an XmlBeanFactory, pass a java.io.InputStream to the constructor. The InputStream will provide the XML to the factory. For example, the following code snippet uses a java.io.FileInputStream to provide a bean definition XML file to XmlBeanFactory.

```
BeanFactory factory = new XmlBeanFactory(new FileInputStream("beans.xml"));
```

To retrieve the bean from a BeanFactory, call the getBean() method by passing the name of the bean you want to retrieve.

```
MyBean myBean = (MyBean) factory.getBean("myBean")
```

67 . What are important ApplicationContext implementations in spring framework?

- ClassPathXmlApplicationContext – This context loads a context definition from an XML file located in the class path, treating context definition files as class path resources.
- FileSystemXmlApplicationContext – This context loads a context definition from an XML file in the filesystem.
- XmlWebApplicationContext – This context loads the context definitions from an XML file contained within a web application.

68 . Explain Bean lifecycle in Spring framework?

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

69 What is bean wiring?

Combining together beans within the Spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together.

70. How do add a bean in spring application?

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
```

```
<beans>
<bean id="foo" class="com.act.Foo"/>
<bean id="bar" class="com.act.Bar"/>
</beans>
```

71. What are singleton beans and how can you create prototype beans?

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named 'singleton' if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

```
<beans>
<bean id="bar" class="com.act.Foo" singleton="false"/>
</beans>
```

72 . What are the important beans lifecycle methods?

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.

73 . How can you override beans default lifecycle methods?

The bean tag has two more important attributes with which you can define your own custom initialization and destroy methods. Here I have shown a small demonstration. Two new methods fooSetup and fooTeardown are to be added to your Foo class.

```
<beans>
<bean id="bar" class="com.act.Foo" init-method="fooSetup" destroy="fooTeardown"/>
</beans>
```

74 . What are Inner Beans?

When wiring beans, if a bean element is embedded to a property tag directly, then that bean is said to the Inner Bean. The drawback of this bean is that it cannot be reused anywhere else.

75. What are the different types of bean injections?

There are two types of bean injections.

- By setter
- By constructor

76. What is Auto wiring?

You can wire the beans as you wish. But spring framework also does this work for you. It can auto wire the related beans together. All you have to do is just set the autowire attribute of bean tag to an autowire type.

```
<beans>
<bean id="bar" class="com.act.Foo" Autowire="autowire type"/>
</beans>
```

77 . What are different types of Autowire types?

There are four different types by which autowiring can be done.

?byName

?byType
?constructor
?autodetect

78. What are the different types of events related to Listeners?
There are a lot of events related to ApplicationContext of spring framework. All the events are subclasses of org.springframework.context.Application-Event. They are

- ContextClosedEvent – This is fired when the context is closed.
- ContextRefreshedEvent – This is fired when the context is initialized or refreshed.
- RequestHandledEvent – This is fired when the web context handles any request.

79. What is an Aspect?
An aspect is the cross-cutting functionality that you are implementing. It is the aspect of your application you are modularizing. An example of an aspect is logging. Logging is something that is required throughout an application. However, because applications tend to be broken down into layers based on functionality, reusing a logging module through inheritance does not make sense. However, you can create a logging aspect and apply it throughout your application using AOP.

80 . What is a Jointpoint?
A joinpoint is a point in the execution of the application where an aspect can be plugged in. This point could be a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect's code can be inserted into the normal flow of your application to add new behavior.

81. What is an Advice?
Advice is the implementation of an aspect. It is something like telling your application of a new behavior. Generally, and advice is inserted into an application at joinpoints.

82. What is a Pointcut?
A pointcut is something that defines at what joinpoints an advice should be applied. Advices can be applied at any joinpoint that is supported by the AOP framework. These Pointcuts allow you to specify where theadvice can be applied.

83. What is an Introduction in AOP?
An introduction allows the user to add new methods or attributes to an existing class. This can then be introduced to an existing class without having to change the structure of the class, but give them the new behavior and state.

84 What is a Target?
A target is the class that is being advised. The class can be a third party class or your own class to which you want to add your own custom behavior. By using the concepts of AOP, the target class is free to center on its major concern, unaware to any advice that is being applied.

85 . What is a Proxy?
A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

86 .What is meant by Weaving?

The process of applying aspects to a target object to create a new proxy object is called as Weaving. The aspects are woven intothe target object at the specified joinpoints.

87 What are the different points where weaving can be applied?

- Compile Time
- Classload Time
- Runtime

88 . What are the different advice types in spring?

- Around : Intercepts the calls to the target method
- Before : This is called before the target method is invoked
- After : This is called after the target method is returned
- Throws : This is called when the target method throws and exception
- Around : org.aopalliance.intercept.MethodInterceptor
- Before : org.springframework.aop.BeforeAdvice
- After : org.springframework.aop.AfterReturningAdvice
- Throws : org.springframework.aop.ThrowsAdvice

89. What are the different types of AutoProxying?

- BeanNameAutoProxyCreator
- DefaultAdvisorAutoProxyCreator
- Metadata autoproxying

90. What kind of exceptions those spring DAO classes throw?

The spring DAO class does not throw any technology related exceptions such as SQLException. They throw exceptions which are subclasses of DataAccessException.

91. What is DataAccessException?

DataAccessException is a RuntimeException. This is an Unchecked Exception. The user is not forced to handle these kinds of exceptions.

92 How can you configure a bean to get DataSource from JNDI?

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName">
<value>java:comp/env/jdbc/myDatasource</value>
</property>
</bean>
```

93. How can you create a DataSource connection pool?

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
<property name="driver">
<value>${db.driver}</value>
</property>
<property name="url">
<value>${db.url}</value>
</property>
<property name="username">
<value>${db.username}</value>
```

```
</property>
<property name="password">
<value>${db.password}</value>
</property>
</bean>
```

94. How JDBC can be used more efficiently in spring framework?

JDBC can be used more efficiently with the help of a template class provided by spring framework called as JdbcTemplate.

95. How JdbcTemplate can be used?

With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database.

JdbcTemplate template = new JdbcTemplate(myDataSource);

A simple DAO class looks like this.

```
public class StudentDaoJdbc implements StudentDao {
        private JdbcTemplate template;

      public void setTemplate(jdbcTemplate) {
         this.jdbcTemplate = jdbcTemplate;
       }
       more..
       }
```

The configuration is shown below.

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource">
<ref bean="dataSource"/>
</property>
</bean>
<bean id="studentDao" class="StudentDaoJdbc">
<property name="jdbcTemplate">
<ref bean="jdbcTemplate"/>
</property>
</bean>
<bean id="courseDao" class="CourseDaoJdbc">
<property name="jdbcTemplate">
<ref bean="jdbcTemplate"/>
</property>
</bean>
```

96. How do you write data to backend in spring using JdbcTemplate?

The JdbcTemplate uses several of these callbacks when writing data to the database. The usefulness you will find in each of these interfaces will vary. There are two simple interfaces. One is PreparedStatementCreator and the other interface is BatchPreparedStatementSetter.

97. Explain about PreparedStatementCreator?

PreparedStatementCreator is one of the most common used interfaces for writing data to

database. The interface has one method createPreparedStatement().

```
PreparedStatement createPreparedStatement(Connection conn)
 throws SQLException;
```

When this interface is implemented, we should create and return a PreparedStatement from the Connection argument, and the exception handling is automatically taken care off. When this interface is implemented, another interface SqlProvider is also implemented which has a method called getSql() which is used to provide sql Strings to JdbcTemplate.

98. Explain about BatchPreparedStatementSetter?

If the user what to update more than one row at a shot then he can go for BatchPreparedStatementSetter. This interface provides two methods

```
setValues(PreparedStatement ps, int i) throws SQLException;
int getBatchSize();
```

The getBatchSize() tells the JdbcTemplate class how many statements to create. And this also determines how many times setValues() will be called.

99. Explain about RowCallbackHandler and why it is used?

In order to navigate through the records we generally go for ResultSet. But spring provides an interface that handles this entire burden and leaves the user to decide what to do with each row. The interface provided by spring is RowCallbackHandler. There is a method processRow() which needs to be implemented so that it is applicable for each and everyrow.

```
void processRow(java.sql.ResultSet rs);
```

100. What is JDBC abstraction and DAO module?

Using this module we can keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. A new layer of meaningful exceptions on top of the error messages given by several database servers is bought in this module. In addition, this module uses Spring's AOP module to provide transaction management services for objects in a Spring application.

Maven in eclipse to use our spring project development:
1-install maven form eclipse plugin--->eclipse market---->maven for eclipse----go to m2E setting--->accept term and conditon --->restart eclipse ide
2-- Create maven project by choosing "quick-start" archetype. file menu---new---maven project---next---choose--org.apache.maven.archetypes-quickstart-1.1 version----next--- Group id :.Company name:nit:---IOCProj14-CollectionInjection finish.
3---Change JRE library to 1.8
4--Add the following <dependency> tags by collecting them from mvnrepository.com
search for spring context
spring context suport

Gradle:
1:-Install build-ship gradle plug-in by eclipse market place and restart eclipse
2:- Create gradle project in eclipse--file menu---new---other---gradle---gradle project--next--project name---finish.
3:-add following dependency script inbuild.gradle for spring context,spring context support jars by collectin them from mvnrepository.com and also enable mavencentral as reposistory.